Instituto Tecnológico de Costa Rica

Área Académica de Ingeniería Mecatrónica

Programa de Licenciatura en Ingeniería Mecatrónica

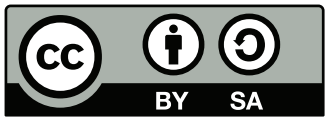
Implementación de capacidades operativas básicas de un robot dedicado a la investigación en robótica cognitiva.

Informe de Trabajo Final de Graduación para optar por el título de

Ingeniero en Mecatrónica con el grado académico de Licenciatura

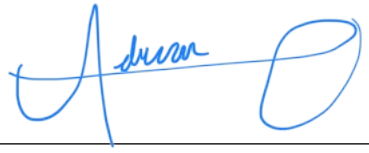
Adrián Ernesto Orozco Rivera

Cartago, 11 de marzo, 2024



Declaro que el presente documento de tesis ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos y resultados experimentales propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de tesis realizado y por el contenido del presente documento.



Adrián Ernesto Orozco Rivera

Cartago, 14 de marzo de 2024

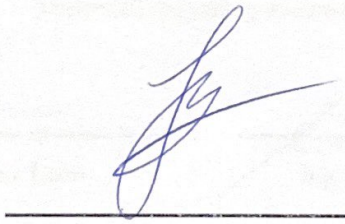
Céd: 1-1735-0311

INSTITUTO TECNOLÓGICO DE COSTA RICA
PROGRAMA DE LICENCIATURA EN INGENIERÍA MECATRÓNICA
PROYECTO FINAL DE GRADUACIÓN
ACTA DE APROBACIÓN

El profesor asesor del presente trabajo final de graduación, indica que el documento presentado por el estudiante cumple con las normas establecidas por el programa de Licenciatura en Ingeniería Mecatrónica del Instituto Tecnológico de Costa Rica para ser defendido ante el jurado evaluador, como requisito final para aprobar el curso Proyecto Final de Graduación y optar así por el título de Ingeniero(a) en Mecatrónica, con el grado académico de Licenciatura.

Estudiante: Adrián Ernesto Orozco Rivera

Proyecto: Implementación de capacidades operativas básicas de un robot dedicado a la investigación en robótica cognitiva



Dr. -Ing. Juan Luis Crespo Mariño

Asesor

Cartago, 11 de Marzo del 2024

INSTITUTO TECNOLÓGICO DE COSTA RICA
PROGRAMA DE LICENCIATURA EN INGENIERÍA MECATRÓNICA
PROYECTO FINAL DE GRADUACIÓN
ACTA DE APROBACIÓN

Proyecto final de graduación defendido ante el presente jurado evaluador como requisito para optar por el título de Ingeniero(a) en Mecatrónica con el grado académico de Licenciatura, según lo establecido por el programa de Licenciatura en Ingeniería Mecatrónica, del Instituto Tecnológico de Costa Rica.

Estudiante: Adrián Ernesto Orozco Rivera

Proyecto: Implementación de capacidades operativas básicas de un robot dedicado a la investigación en robótica cognitiva

Miembros del jurado evaluador

Dra. -Ing. Gabriela Ortiz León
Jurado

Ing. Eduardo José Asenjo Castillo
Jurado

Los miembros de este jurado dan fe de que el presente proyecto final de graduación ha sido aprobado y cumple con las normas establecidas por el programa de Licenciatura en Ingeniería Mecatrónica.

Cartago, 11 de Marzo del 2024

Resumen

Se ejecuta el desarrollo de un conjunto de capacidades de operación básicas conocidas como acciones atómicas en un ambiente de robótica cognitiva de manipuladores para el laboratorio LIANA *(Laboratorio de Inteligencia Artificial para las Ciencias Naturales)* del Instituto Tecnológico de Costa Rica. Para solucionar el problema se hace uso de una metodología de diseño de ingeniería para analizar las necesidades del diseño y implementar un sistema con especificaciones que determinan el rendimiento de la solución. Se ejecutan pruebas que permiten validar si los resultados son positivos. Los resultados se analizan y se procede a concluir sobre el cumplimiento de las necesidades del cliente.

Palabras clave: ROS, nodos, eslabones, articulaciones, manipulador

Abstract

This project documents the development of a set of basic operational capabilities, known as atomic actions. It is conducted within a cognitive robotics environment for manipulators at the *LIANA* Laboratory (Laboratory of Artificial Intelligence for Natural Sciences) of the Instituto Tecnológico de Costa Rica. To address the problem, an engineering design methodology is employed to analyze the design requirements and implement a system with specifications that determine the solution's performance. Tests are carried out to validate whether the results are positive. The results are analyzed, and conclusions are drawn regarding the fulfillment of the client's needs.

Keywords: ROS, nodes, links, joints, manipulator

a mi familia y amigos

Agradecimientos

Agradezco a mi madre Ana y padre Alberto que me dieron la vida y invirtieron muchos años en criarme para que siga mis sueños y encuentre mi lugar en la sociedad. Gracias a mi hermano Elias por siempre darme su buen ejemplo de vida.

Gracias a mis abuelas y abuelos, que me enseñaron a apreciar la simpleza de la vida y saborear los momentos de paz. Gracias a mi madrina Hilda que siempre me impulsó en mis estudios y me prestó el carro Palomo para poder ir al TEC.

Gracias a mi novia Daniela porque me apoyaste en los momentos en los que más dudé de mi mismo y el valor de mi trabajo como ingeniero y persona.

Agradezco a los profesores Juan Luis Crespo Mariño y Esteban Arias Méndez por formar parte de este proceso tan importante en mi vida. Espero haber cumplido con las expectativas que tenían para mi proyecto en el laboratorio LIANA.

Gracias al TEC y a todos los profesores que me compartieron conocimiento durante todos estos años.

Por último, agradezco a la vida, porque es una aventura. Espero que la ingeniería expanda mis horizontes.

Adrián Ernesto Orozco Rivera

Cartago, 14 de marzo de 2024

Índice general

Índice de figuras

Índice de tablas

Listings

Capítulo 1

Introducción

1.1. Entorno del proyecto

Este proyecto se realiza en el Laboratorio de Inteligencia Artificial para las Ciencias Naturales *(LIANA)*. Esta es una unidad de investigación del Área Académica de Ingeniería Mecatrónica del Instituto Tecnológico de Costa Rica. Los objetivos generales de los proyectos buscan el desarrollo de soluciones computacionales inteligentes que puedan ser aplicadas en las ciencias naturales. Además, la investigación ejecutada en el laboratorio busca la generación de conocimiento y habilidades de administración de proyectos en sus miembros [1].

El entorno del proyecto es en investigación de robótica cognitiva. Actualmente, el laboratorio ha ejecutado tres proyectos en esta área previamente. El primer proyecto es la selección del entorno de robótica cognitiva de interés para el laboratorio desarrollado por Emmanuel Fallas Hernandez [2]. Es una arquitectura cognitiva nombrada Open Source Cognitive Applied Robot *OSCAR*. El segundo proyecto es la construcción del robot cognitivo *OSCAR* y inclusión de sensores desarrollado por Alejandro Gomez Vega [3]. El tercer proyecto es el rediseño del robot *OSCAR* para restaurar su movimiento y es realizado por María Fernanda Araya Soto [4]. Como cuarto proyecto en el área de robótica cognitiva se propone el desarrollo de capacidades de operación básicas conocidas como acciones atómicas para controlar sistemas robóticos y hacer experimentos de laboratorio.

1.2. Definición del problema

En este documento se describe el proceso de diseño de una solución a un problema de índole mecatrónico. Se aborda la necesidad de desarrollar una investigación en tecnologías de software modernas que permitan capacidades de operación básicas en un ambiente de robótica cognitiva. El laboratorio cuenta con la arquitectura cognitiva *OSCAR* pero su kit de desarrollo de software pronto quedará deprecado. También, el modelo físico del

robot no se puede utilizar debido a que requiere una reconstrucción. Debido a las razones previas, se plantea ejecutar este proyecto que consistirá en plantear una arquitectura cognitiva basada en *OSCAR* pero que esté desarrollada en un ambiente moderno. El laboratorio necesita una base de datos actualizada en cuanto a la implementación de sistemas robóticos cognitivos.

1.2.1. Riesgos asociados al problema

Abordar el desarrollo de esta solución implica riesgos que se deben reconocer desde su planteamiento. El primero de estos riesgos corresponde a la situación actual de los brazos robóticos *OSCAR*. Durante la etapa de prototipado, se detectan problemas de movimiento y control. Debido a esta razón, Fernanda Araya Trejos rediseña los brazos para mejorar el rendimiento de operación [4]. No obstante, el costo de los componentes es alto y su financiamiento es condicional. Además el proyecto de rediseño se ejecuta durante el primer semestre del 2023 pero requiere una prórroga y su defensa se lleva a cabo el 31 de Agosto del 2023. Este proyecto se desarrolla el segundo semestre del 2023 y coincide en el desarrollo del rediseño. En resumen, hay un riesgo de costo y de tiempo asociado a este proyecto. Es una posibilidad que el robot *OSCAR* no esté reconstruido a tiempo para ser utilizado en la validación de las acciones atómicas que se desarrollan en este proyecto.

Los riesgos previamente descritos ocasionan que se tenga que plantear un plan de remedio para poder ejecutar la investigación sin tener que depender de la situación del modelo físico del robot. La principal consecuencia del plan es en el método de validación para las capacidades de operación que se desarrollan. Debido a que el robot no se reconstruye a tiempo para la realización de este proyecto se opta por utilizar un software de simulación de robótica.

1.3. Síntesis del problema

En síntesis, el laboratorio necesita continuar con el desarrollo de la línea de investigación en robótica cognitiva. Se debe desarrollar un conjunto de acciones atómicas que permitan implementar capacidades de operación básicas usando tecnología moderna. Se debe plantear una estrategia de remedio para lidiar con los riesgos planteados que limitan el método de validación y el tiempo de desarrollo. La primera posibilidad es el uso de *OSCAR* rediseño si se logra reconstruir a tiempo. La segunda posibilidad es usar un ambiente simulado de un manipulador similar a *OSCAR* para realizar la validación. Es importante mencionar que la simulación es una opción válida ya que hay simuladores de física que permiten el estudio de movimiento de ensambles robóticos.

1.4. Objetivos

1.4.1. Objetivo General

Diseñar un conjunto de acciones atómicas de movimiento para un robot manipulador de modo que sirvan para la realización de experimentos en el área de robótica cognitiva.

1.4.2. Objetivos Específicos

1. Determinar las especificaciones requeridas para diseñar el conjunto de acciones atómicas del robot cognitivo de acuerdo con criterio bibliográfico y criterio experto del laboratorio LIANA.
    - Entregable: Lista de especificaciones en base a las necesidades del cliente.
    - Indicador: Valores objetivo de las especificaciones.
2. Diseñar un conjunto de acciones atómicas de forma que sean compatibles con el lenguaje de programación *ROS* 2.
    - Entregable: Código de acciones acciones atómicas en ROS 2.
    - Indicador: Video demostrativo de ejecución de cada una de las acciones de forma correcta en el robot cumpliendo con los valores objetivo de las especificaciones.
3. Desarrollar un ambiente de simulación de robótica para la validación de las acciones atómicas.
    - Entregable: Ambiente de simulación de robótica cognitiva.
    - Indicador: Funcionamiento del robot en el ambiente simulado siguiendo las especificaciones de las acciones atómicas
4. Validar el funcionamiento de las acciones atómicas por medio de experimentos que utilicen el conjunto de acciones en el ambiente de simulación de robotica cognitiva.
    - Entregable: Conjunto de pruebas para validación de ambiente de robótica.
    - Indicador: Demostración de cumplimiento de especificaciones por medio de la medición de los resultados de las pruebas realizadas.

Capítulo 2

Marco teórico

En este capítulo se detallan los fundamentos teóricos necesarios para el desarrollo de la solución. Los temas de mayor interés son:

- Robótica
- Kit de desarrollo de software *ROS* 2
- Ambiente de simulación *Gazebo*
- Herramientas de software

2.1. Robótica

En esta sección se exploran los conceptos necesarios para entender el robot que se utiliza en la simulación de este proyecto. Los conceptos son centrados en robótica de manipuladores.

2.1.1. Ambiente de Robótica *OSCAR*

En el contexto del desarrollo y la investigación en robótica, OSCAR representa un esfuerzo colaborativo e interdisciplinario destinado a fomentar el avance en el campo de la robótica cognitiva [2] [3] [4]. Este entorno, situado en el Laboratorio de Inteligencia Artificial para las Ciencias Naturales (LIANA) del Instituto Tecnológico de Costa Rica, es instrumento para el desarrollo de proyectos. La plataforma OSCAR, compuesta por manipuladores robóticos THOR, no solo facilita la exploración de nuevas ideas y tecnologías sino que también sirve como un recurso educativo invaluable para estudiantes e investigadores.

2.1.2. Manipulador robótico

Un manipulador robótico, comúnmente conocido como brazo robótico, es una clase de robot altamente versátil y adaptable diseñado para replicar, hasta cierto punto, la funcionalidad y los movimientos del brazo humano. Estos dispositivos están compuestos por una serie de segmentos interconectados, denominados eslabones, unidos por articulaciones que pueden ser rotacionales o prismáticas, permitiendo así una amplia gama de movimientos [5].

Estructura y componentes

- Articulación (joint): Una articulación en un manipulador robótico es el elemento que permite el movimiento relativo entre dos eslabones adyacentes. Actúa como el punto de conexión y permite diversos tipos de movimientos, dependiendo de su diseño.
- Eslabón (link): Un eslabón o segmento en un manipulador robótico es un cuerpo rígido que conecta dos articulaciones. Los eslabones son los componentes estructurales que forman el .^{es}queleto"del manipulador, extendiéndose desde la base fija hasta el efector final. Cada eslabón contribuye a la estructura general del manipulador y puede alojar componentes adicionales como cables, actuadores o sensores.
- Efector Final: El efector final, o herramienta, es el componente del manipulador que interactúa directamente con el entorno para realizar una tarea específica. Puede variar desde pinzas simples hasta herramientas más complejas diseñadas para operaciones específicas como soldadura, pintura o ensamblaje.

Tipos de articulaciones

Mendoza [6] describe los distintos tipos de articulaciones que se utilizan en robótica.

- Prismática: Esta compuesta por dos uniones que se desplazan a lo largo de cada una generando un movimiento en línea recta.
- Rotacional: Generan movimiento rotativo sobre un único eje.
- Cilíndrica: Permite movimiento prismático y rotacional.
- Planar: Para movimientos en un plano.
- Esférica: Combina 3 giros en 3 direcciones perpendiculares.

El ambiente de robótica *OSCAR* y los manipuladores en general utilizan articulaciones **rotacionales** principalmente.

Tipos de movimiento de rotacionales

En la robótica, hay tres tipos básicos de movimientos básicos de rotación [7]:

1. Roll: Este movimiento rota en torno al eje x de un marco de referencia.
2. Pitch: La rotación se da en torno al eje y del marco de referencia.
3. Yaw: El giro de la articulación se da en torno al eje z del marco de referencia.

Grados de libertad

Los grados de libertad (GDL) en robótica se refieren al número de ejes independientes en los que un robot puede moverse. Estos incluyen traslaciones y rotaciones en el espacio tridimensional, permitiendo al robot alcanzar una posición y orientación específicas. Cada grado de libertad representa una dirección única en la que el robot puede moverse o girar, lo que afecta directamente su capacidad para realizar tareas complejas [8].

2.2. Kit de desarrollo de software *ROS*

Robot Operating System *ROS* es un kit de desarrollo de software que proporciona un conjunto de herramientas y bibliotecas de código abierto. Este ambiente de desarrollo es ampliamente usado para la implementación de robots. Ofrece infraestructura de software flexible y distribuida. Las principales necesidades que este kit ayuda a resolver son:

- Control de hardware: Se pueden plantear estrategias de control de los distintos actuadores de un robot para ejecutar movimientos y acciones.
- Percepción de objetos: Los robots pueden tomar decisiones en base a las condiciones de su ambiente. Es posible implementar sistemas de visión artificial para percibir objetos en el espacio de trabajo. Además, se puede agregar sensores que permiten el flujo de información del ambiente al sistema.
- Planificación de movimiento: Hay bibliotecas disponibles que permiten calcular trayectorias seguras y eficaces.
- Simulación de ambientes robóticos: La validación del funcionamiento de los robots se puede simular. De esta forma se puede estudiar si hay problemas mecánicos sin tener que arriesgar los modelos físicos.
- Escalabilidad y modularidad: *ROS* permite implementar sistemas robóticos de múltiples agentes que colaboran entre si.

En general, este ambiente ofrece muchas capacidades útiles que permiten el diseño de robots desde su conceptualización hasta su validación. La Fig. 2.1 muestra una simulación de ejemplo de un ambiente robótico. Se observa que se coloca un modelo de un robot en un espacio de trabajo. Además, se implementan las piezas STL de un robot comercial lo cuál habilita una visualización fiel a la realidad [9].

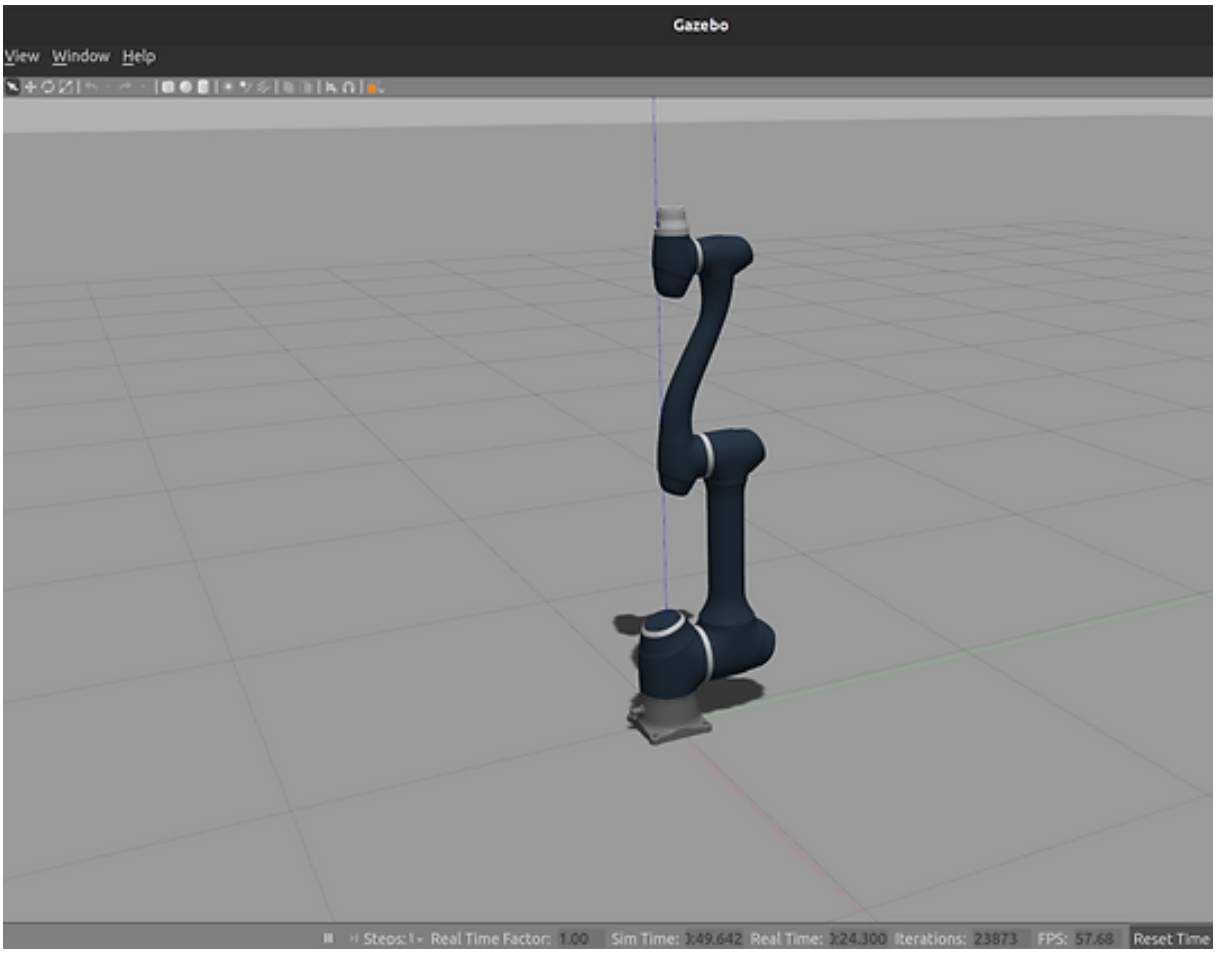


Figura 2.1: Ejemplo de utilización de *ROS* [9]

Existen 2 versiones de *ROS*. *ROS* 1 se desarrolla con fines académicos. No obstante, a partir del 2025 quedará obsoleto. La segunda versión *ROS* 2 se desarrolla para robustecer la flexibilidad del sistema y poder implementarlo en soluciones comerciales. A largo plazo, *ROS* 2 es la mejor opción para desarrollar ambientes robóticos debido al soporte que se planea dar a esta plataforma [10]. El enfoque de este apartado del marco teórico será en *ROS* 2.

2.2.1. Arquitectura de *ROS* 2

La arquitectura de *ROS* permite integraciones a tiempo real. Es decir, se pueden obtener mediciones de sensores y tomar decisiones en base a la percepción obtenida del ambiente. Se puede implementar controladores para los actuadores del robot. También cuenta

con capacidades de programación de algoritmos IoT en lenguajes de programación como Python y C++. Todos los componentes previos se conocen como nodos. Dichos nodos se encuentran distribuidos en capas que se comunican entre si [10]. Las capas principales son las siguientes:

- Comunicación: Los nodos de esta se encargan de transmitir mensajes entre el robot y su espacio de trabajo.
- Control y planificación: En esta capa se plantean nodos de planificación de trayectorias.
- Percepción: Son los nodos que representan el espacio de trabajo del robot y los objetos presentes.
- Sensores: En esta capa se colocan los nodos que permiten obtener información acerca del espacio de trabajo.
- Actuación: El robot se representa con los nodos de esta capa.

En la Fig. 2.2 se muestra un ejemplo de distribución de un ambiente robótico en *ROS* 2.

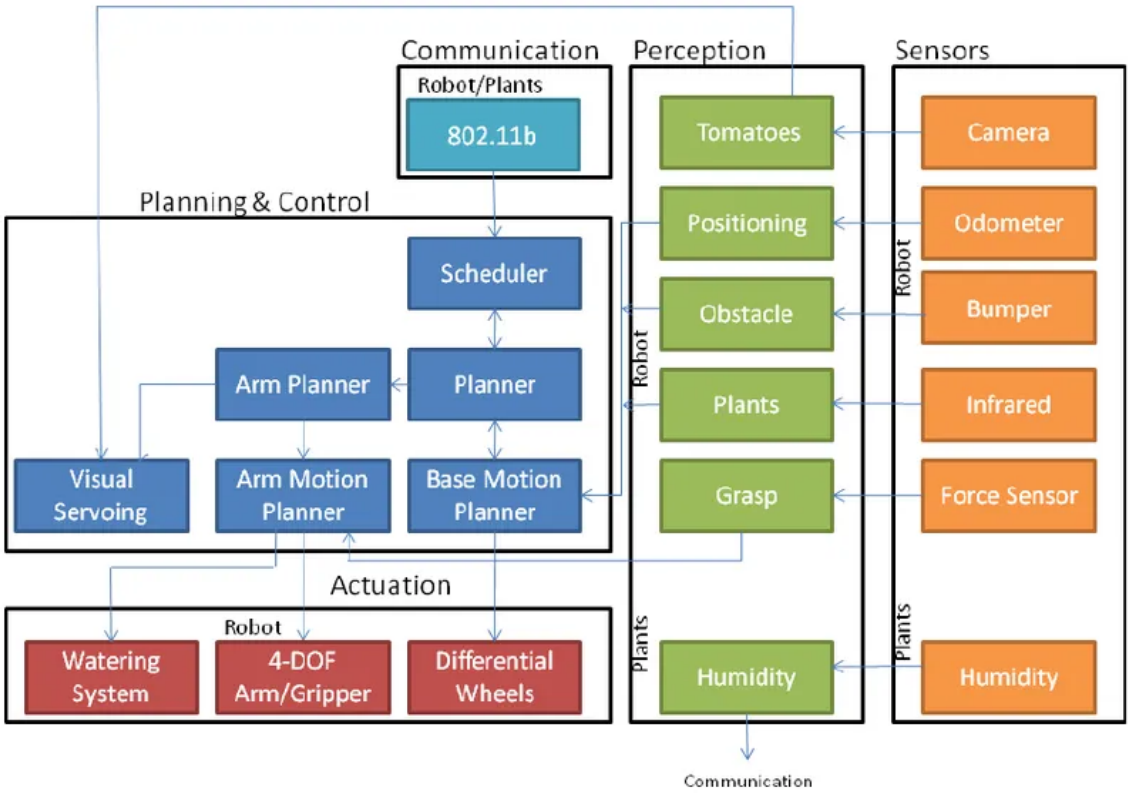


Figura 2.2: Diagrama de distribución de nodos en arquitectura de *ROS* 2 [10]

2.2.2. Estructura de comunicación

Para lograr que las aplicaciones desarrolladas en *ROS* 2 funcionen se debe comprender la estructura de comunicación. La Fig. 2.3 muestra la forma en la cual se comunican los nodos entre si. Por medio de tópicos se envían mensajes comunicando información del sistema. Los nodos pueden funcionar como publicadores y como subscriptores, es decir enviar o recibir información. Los servicios se encargan de coordinar las acciones de los nodos [10].

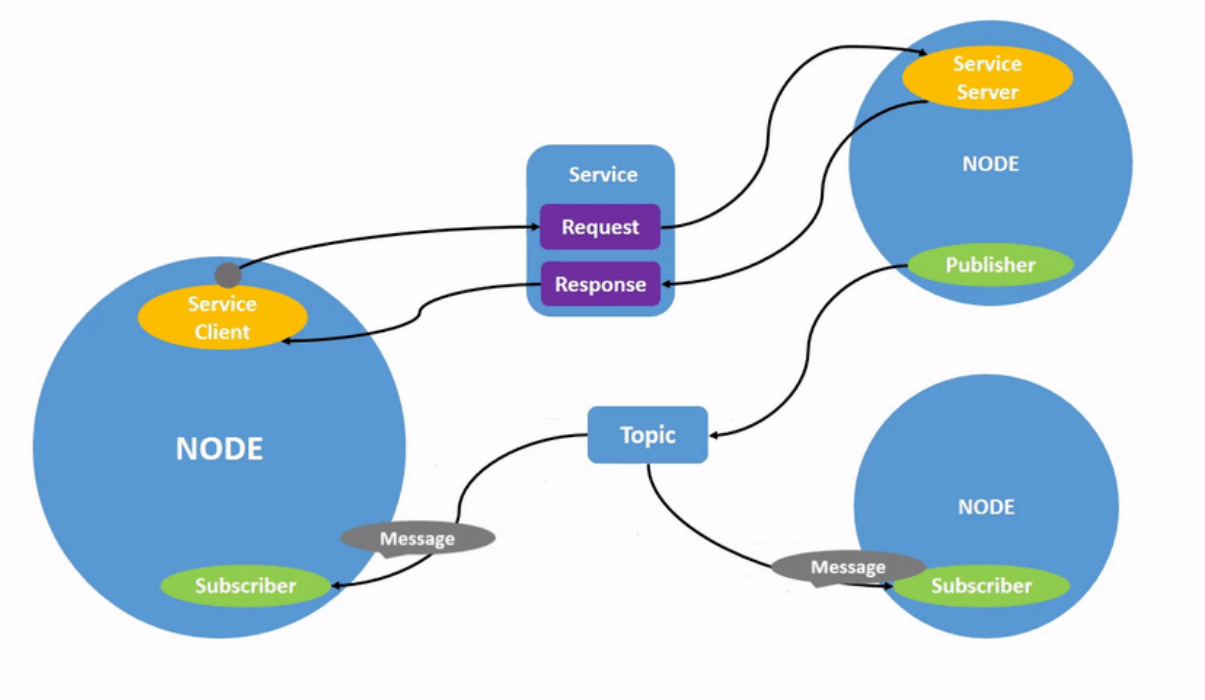


Figura 2.3: Diagrama de comunicación de nodos en arquitectura de *ROS* 2 [10]

Ahora, cada uno de los elementos se define formalmente.

Nodo

Un nodo en ROS 2 es una entidad que realiza cálculos. Los nodos pueden publicar o suscribirse a un tópico para recibir o enviar mensajes. Los nodos pueden comunicarse entre sí para realizar tareas complejas dentro de una red de ROS 2. Los nodos son fundamentales en la arquitectura de ROS 2 ya que permiten la modularidad y la reutilización del código [11].

Mensaje

Un mensaje en ROS 2 es una estructura de datos, definida en un formato específico, que se utiliza para la comunicación entre nodos. Los mensajes pueden contener tipos de datos

primitivos como enteros, flotantes, cadenas de texto y también estructuras más complejas como arreglos o incluso otros mensajes [12].

Tópico

Un tópico en ROS 2 es un canal de nombres que se utiliza para la comunicación entre nodos a través de mensajes. Los nodos pueden publicar mensajes en un tópico o suscribirse a un tópico para recibir mensajes. La comunicación a través de tópicos permite la decoupling entre los productores y consumidores de mensajes [13].

Servicio

Un servicio en ROS 2 es un mecanismo de comunicación que permite la interacción síncrona entre nodos, siguiendo un patrón de solicitud y respuesta. A diferencia de los tópicos, que son adecuados para la transmisión de datos continua, un servicio es utilizado cuando un nodo necesita realizar una petición específica a otro nodo y esperar una respuesta. Esta comunicación es particularmente útil para tareas que requieren una acción inmediata o un cálculo basado en una solicitud específica [14].

Acciones

Las acciones en ROS 2 son una forma de comunicación entre nodos diseñada para facilitar la ejecución de tareas a largo plazo que pueden requerir interacción durante su ejecución. A diferencia de los servicios, que son ideales para solicitudes síncronas de solicitud y respuesta, las acciones permiten enviar una solicitud y luego proporcionar feedback, recibir el resultado de la acción o incluso cancelar la acción mientras se está ejecutando [15].

Paquete

Los paquetes en ROS 2 son la unidad principal de organización del software en el ecosistema de ROS. Un paquete puede contener nodos de ROS, bibliotecas, datasets, archivos de configuración, o cualquier otra cosa que constituya un módulo de software útil. Los paquetes están diseñados para ser fácilmente compartibles y reutilizables en diferentes proyectos de ROS [16].

2.2.3. Formato Unificado de Descripción de Robots (URDF)

El Formato Unificado de Descripción de Robots (URDF, por sus siglas en inglés) es un formato basado en XML utilizado para describir la estructura física y visual de un robot en el ecosistema de ROS (Robot Operating System). El URDF define los componentes de un robot, como enlaces (links), articulaciones (joints), sensores, actuadores y otras

propiedades físicas, de manera que el sistema ROS pueda entender y manipular el modelo del robot.

Componentes Principales del URDF

Los siguientes conceptos detallados en la documentación de *ROS* 2 [17] son clave para la descripción de modelos físicos en el ambiente simulado:

- Enlaces (Links): Representan las partes rígidas del robot, como el chasis, los brazos, las piernas o las ruedas. Cada enlace se define por sus propiedades visuales, de colisión y de inercia.
- Articulaciones (Joints): Definen las relaciones cinemáticas entre los enlaces. Las articulaciones pueden ser de varios tipos, como rotacionales, prismáticas (para movimientos lineales) y fijas, dictando cómo un enlace puede moverse con respecto a otro.
- Sensores y Actuadores: Aunque el URDF está principalmente enfocado en la estructura física, se pueden incorporar extensiones para describir sensores y actuadores, lo que facilita la integración con la simulación y el control en ROS.

Integración del URDF en ROS y Gazebo

En ROS, el URDF se utiliza para diversas tareas, como la planificación del movimiento, la simulación de la dinámica, la visualización en RViz y la integración con herramientas de simulación como Gazebo. El URDF permite a los desarrolladores y sistemas ROS entender la estructura y las capacidades de movimiento de un robot, facilitando la programación y control de robots complejos.

Gazebo, por otro lado, aprovecha el URDF para importar modelos de robots a su entorno de simulación. Aunque Gazebo tiene su propio formato de descripción (SDF) [17], la compatibilidad con URDF permite a los usuarios de ROS llevar sus modelos existentes a Gazebo sin necesidad de conversiones complejas, facilitando la simulación realista de robots en entornos virtuales detallados.

Creación y Uso del URDF

Crear un archivo URDF requiere definir la estructura del robot en XML, especificando cada enlace y articulación con sus respectivas propiedades. Herramientas como xacro (XML Macros) permiten simplificar y modularizar la creación de archivos URDF, haciéndolos más manejables y reutilizables. Una vez definido, el URDF se puede cargar en ROS y Gazebo para simulación, visualización y desarrollo de algoritmos de control [18].

2.2.4. Cinemática robótica con *MoveIt*

La cinemática robótica es una rama esencial de la robótica que se ocupa del estudio del movimiento de los robots sin considerar las fuerzas que lo provocan. Esta área se centra en describir, de manera matemática y geométrica, la posición, velocidad y aceleración de los componentes de un robot, especialmente en lo que respecta a sus articulaciones y el efector final. A continuación se explican dos conceptos importantes

Cinemática directa

La cinemática directa, o análisis de cinemática directa, se refiere al cálculo de la posición y orientación del efector final de un robot, dado un conjunto conocido de valores de las variables articulares. Este proceso implica el uso de las especificaciones geométricas de cada eslabón del robot para desarrollar ecuaciones que describan la posición del efector final en el espacio. La cinemática directa es fundamental para la planificación de trayectorias y la programación de tareas robóticas.

Cinemática inversa

La cinemática inversa aborda el problema opuesto al de la cinemática directa. Se enfoca en determinar los valores necesarios de las variables articulares para que el efector final del robot alcance una posición y orientación específicas en el espacio. Este problema es más complejo que la cinemática directa y a menudo presenta múltiples soluciones, ninguna solución o requiere métodos numéricos para su resolución. La cinemática inversa es crucial para el control reactivo y la adaptación a entornos dinámicos en robótica. En el contexto de *ROS* 2, la cinemática de los robots se resuelve usando el paquete de *MoveIt*. Es un planificador de trayectorias que se encarga de recibir una descripción del montaje de las articulaciones del robot, sus límites físicos y sus relaciones de movimiento para calcular si las poses o trayectorias solicitadas se pueden alcanzar sin causar alguna colisión o perturbación de los límites físicos [19].

La principal ventaja de la plataforma *ROS* 2 es que muchos problemas esenciales de robots manipuladores ya han sido documentados por la comunidad de robótica. Los paquetes se encargan de resolver complejas ecuaciones matriciales recibiendo como parámetros la pose del robot en términos de *roll*, *pitch*, *yaw*. El sistema es capaz de reducir la carga de computación convirtiendo la pose en un cuaternión.

Un cuaternión es una representación de la orientación de las articulaciones de un robot fácil de procesar de forma computacional porque reduce de forma considerable el tamaño de las operaciones matriciales que se deben realizar. Es una forma alternativa de representar una pose de ángulos de euler *roll*, *pitch* y *yaw*. Por esta razón, es un recurso valioso para sistemas robóticos que requieren bajos tiempos de respuesta [20].

2.3. Ambiente de simulación *Gazebo*

Gazebo es un simulador 3D de código abierto ampliamente utilizado en la comunidad de robótica para simular complejos entornos y robots en un espacio tridimensional. Permite a los desarrolladores probar algoritmos, diseñar robots y entrenar sistemas de inteligencia artificial en entornos virtuales realistas. Gazebo ofrece una robusta física de simulación, una amplia biblioteca de modelos, y es capaz de simular poblaciones de robots en entornos complejos y dinámicos. Es compatible con ROS (Robot Operating System), lo que facilita la integración de simulaciones con código de control de robots reales y virtuales [21].

2.3.1. Simulación de Física Avanzada

Gazebo integra varios motores de física de alto rendimiento como ODE, Bullet, Simbody y DART, permitiendo una simulación precisa de la dinámica de cuerpos rígidos, colisiones y otras interacciones físicas. La posibilidad de elegir entre diferentes motores de física proporciona una flexibilidad significativa, permitiendo a los usuarios optimizar sus simulaciones para diversas necesidades. Koenig y Howard destacan la importancia de esta flexibilidad en la simulación de entornos robóticos complejos [22].

2.3.2. Entornos y Modelos Detallados

Gazebo proporciona una extensa biblioteca de modelos y entornos que van desde simples componentes robóticos hasta paisajes urbanos detallados. Esta biblioteca facilita el prototipado y la simulación en una amplia gama de escenarios sin necesidad de construir físicamente los entornos. Los usuarios también tienen la capacidad de diseñar modelos personalizados, lo que amplía aún más la aplicabilidad de Gazebo en la investigación y desarrollo. En la Fig. 2.4 se muestra un ejemplo donde un robot debe seguir a una persona. Se aprecia que el entorno de trabajo del robot permite un ajuste cercano a la realidad.

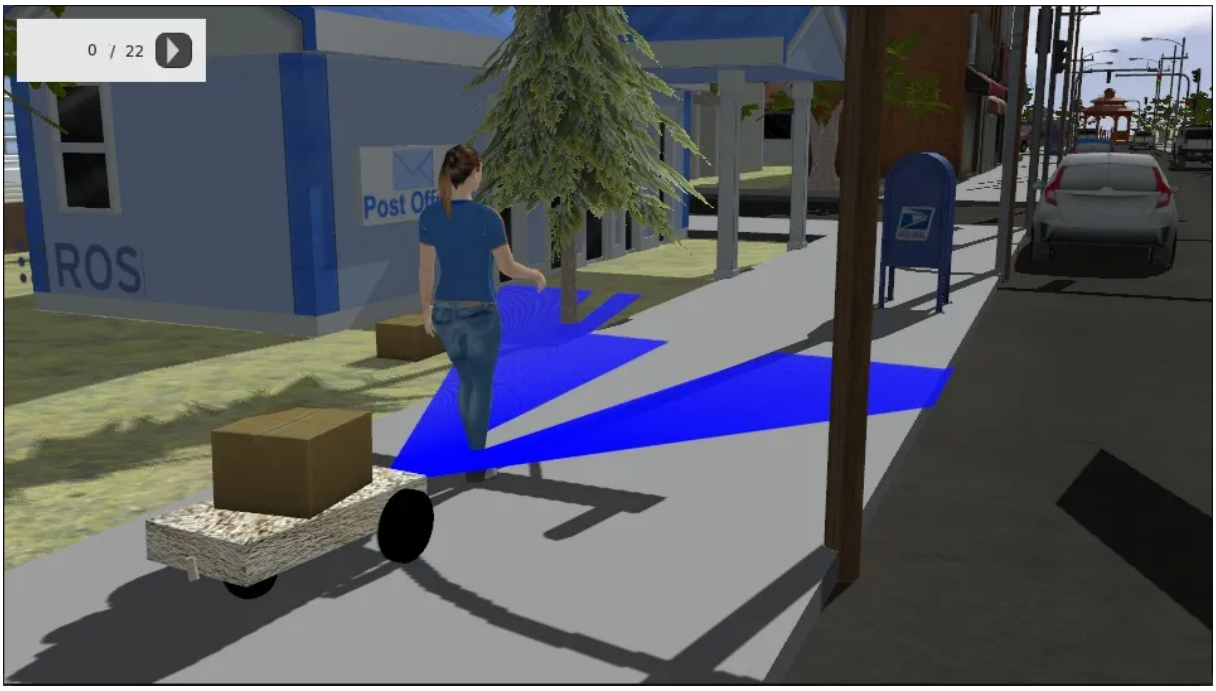


Figura 2.4: Ejemplo de ambiente de simulación complejo en Gazebo [21]

2.3.3. Integración con ROS

La compatibilidad de Gazebo con ROS 2 permite una transición sin problemas de la simulación al mundo real, haciendo posible que los algoritmos desarrollados y probados en Gazebo se implementen directamente en robots reales. Esta integración es fundamental para el desarrollo iterativo y la prueba de sistemas robóticos, facilitando la experimentación y el ajuste fino de algoritmos en un entorno controlado antes de la implementación física [23].

2.3.4. Interfaz Gráfica y Herramientas de Desarrollo

La interfaz gráfica de usuario de Gazebo y sus herramientas de desarrollo permiten una interacción intuitiva con la simulación, facilitando el diseño, la implementación y la depuración de modelos robóticos. Esta accesibilidad es crucial para investigadores, ingenieros y educadores, permitiendo un enfoque más eficiente y efectivo en el desarrollo de sistemas robóticos complejos [24]. En la Fig. 2.5 se muestra que la interfaz cuenta con un sistema de anidamiento de componentes que permite administrar los recursos presentes en el ambiente robótico.

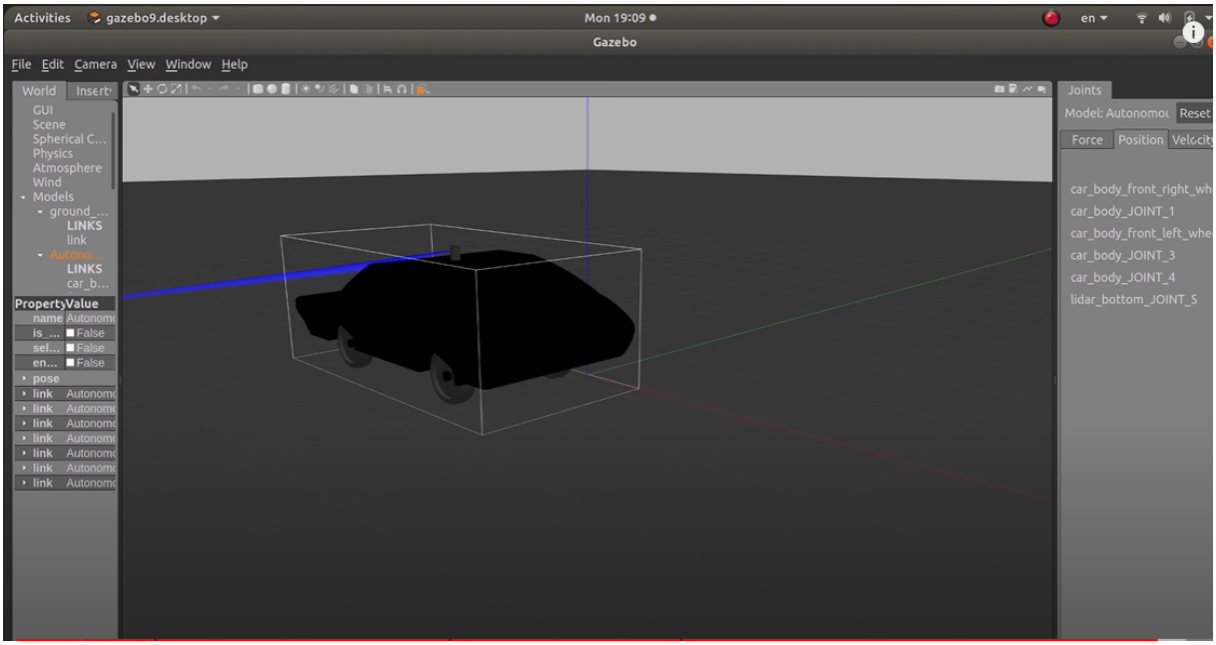


Figura 2.5: Interfaz gráfica de Gazebo [24]

2.3.5. Soporte de Sensores y Actuadores

Gazebo simula una amplia gama de sensores y actuadores, lo que permite a los desarrolladores probar y validar sistemas de percepción y control en entornos virtuales. Esta capacidad es esencial para implementar sistemas robóticos que interactúan de manera efectiva con el mundo real, proporcionando un medio para evaluar el rendimiento del sistema en diversas condiciones sin los riesgos o costos asociados con las pruebas físicas [25]. En la Fig. 2.6 se muestra la implementación de una cámara en un ambiente robótico de ejemplo.

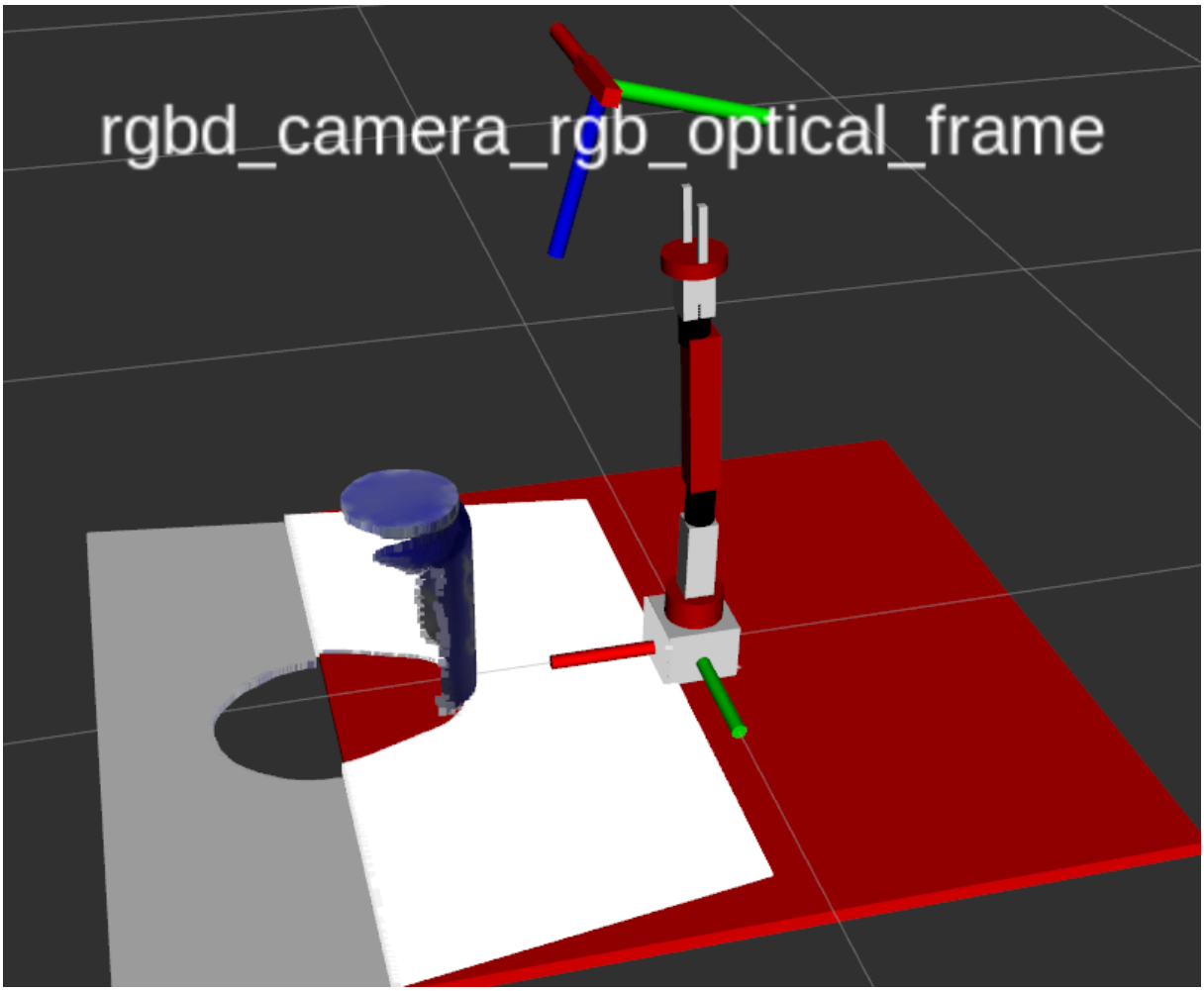


Figura 2.6: Ejemplo de simulación de sensores en Gazebo [26]

2.4. Herramientas de software

Hay conceptos importantes relacionados a herramientas de software que se usan para la implementación de este proyecto. A continuación se describen los conceptos más importantes.

2.4.1. Interfaz de programación de aplicaciones

Una interfaz de programación de aplicaciones, o *API*, es una herramienta que permite habilitar la comunicación entre aplicaciones de forma sencilla. Es una poderosa herramienta porque se puede comunicar información por medio del internet y proveer servicios de aplicaciones a desarrolladores de forma sencilla. Un *API* tiene una estructura de mensajería de solicitudes y respuestas. Permiten externalizar servicios complejos como el reconocimiento de comandos de voz a otras plataformas que resuelven este problema [27].

Ngrok es una herramienta que permite a los desarrolladores exponer un servidor web local

a Internet. Esto se logra creando un túnel desde un subdominio público en el servidor de *Ngrok* hacia el servidor web local del desarrollador. La principal utilidad de *Ngrok* es facilitar la compartición, prueba y depuración de aplicaciones API locales con otras personas, sin necesidad de desplegar la aplicación en un servidor de producción [28].

2.4.2. Asistente de voz Alexa

Alexa es una tecnología desarrollada por *Amazon*. Es un modelo de reconocimiento de lenguaje que tiene capacidades de asistente de voz. Este sistema ofrece un *API* que permite integrar las capacidades de reconocimiento en aplicaciones de terceros. Este tipo de tecnología es innovadora porque permite ejecutar programas de computación por medio de comandos de voz. Las posibilidades de integraciones son amplias [29]. El funcionamiento básico del asistente de voz es el siguiente:

1. Activación: Generalmente, el asistente se activa mediante una palabra clave o palabra de activación específica, como *Alexa* en el caso de los dispositivos Echo de Amazon.

2. Captura de voz: Una vez activado, el asistente escucha y graba la solicitud verbal del usuario.

3. Procesamiento y respuesta: La grabación se envía a los servidores de la nube, donde se procesa utilizando algoritmos avanzados para interpretar la solicitud y generar una respuesta adecuada.

4. Acción: El asistente realiza la acción solicitada, que puede ser proporcionar una respuesta verbal, reproducir contenido multimedia, enviar información a otro dispositivo, entre otras opciones.

Capítulo 3

Desarrollo de la propuesta de solución

En este capítulo se desglosa el proceso de diseño de la solución en base a una metodología de proyectos de ingeniería. Se cubren los aspectos teóricos necesarios para desarrollar la solución y los criterios que respaldan la validez del resultado.

3.1. Metodología

Este proyecto utiliza como base de desarrollo la metodología de diseño de ingeniería de Ulrich & Eppinger (UE). Esta metodología se usa para el desarrollo de productos y tiene un flujo de trabajo iterativo que permite iterar en cualquier etapa del proceso para realizar correcciones [30]. En la Fig. 3.1 se muestran las etapas que se ejecutan para obtener la solución a la problemática. Sin embargo, es importante mencionar que se usa una versión modificada de la metodología ajustada a las necesidades del proyecto. La modificación principal es que se realiza el análisis económico luego de determinar las necesidades y especificaciones del proyecto y también una vez se completa el diseño de la solución. El análisis económico se divide en 2 partes. Esta elección se realiza para ajustarse a las necesidades específicas del proyecto y cliente.

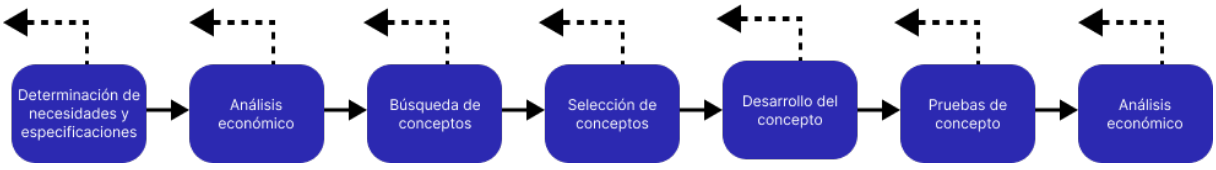


Figura 3.1: Metodología de diseño Ulrich & Eppinger. Elaboración propia: Figma

3.2. Desarrollo de la solución

A continuación se desglosa el proceso y razonamiento aplicado en cada etapa para generar soluciones que cumplen con las necesidades del proyecto. Por medio de las etapas de diseño

se obtendrá como resultado distintas opciones que son capaces de resolver la problemática central. Además, por medio del flujo de trabajo se pueden comparar las soluciones para determinar de forma cuantitativa el concepto de solución ganador. No obstante, es importante mencionar que se aplican modificaciones en la metodología Ulrich & Eppinger (UE) [30]. Dichos cambios son descritos en las etapas correspondientes.

3.2.1. Entrevista con el cliente

El tres de agosto del 2023 se lleva a cabo una reunión con el cliente para definir las necesidades que deben resolverse en este trabajo. El cliente de este proyecto es Esteban Arias Mendez, profesor de la escuela de computación y miembro de *LIANA*. También participa Juan Luis Crespo Mariño, profesor del área académica de ingeniería mecatrónica y director del laboratorio. Previo a esta reunión se preparan preguntas en base a la descripción general recibida. A continuación se desglosa el listado:

1. ¿Cuál es el estado actual de *OSCAR* y la línea de investigación de robótica cognitiva del laboratorio?
2. ¿Qué es una acción atómica?
3. ¿Cuáles acciones atómicas se necesitan para OSCAR?
4. ¿Cuáles son los parámetros que deben variar en las acciones atómicas?
5. ¿Se debe implementar capacidades de visión artificial en OSCAR?
6. ¿Cuál ambiente de desarrollo es el requerido para la solución?

Para la primer pregunta, el cliente indica que la situación de *OSCAR* es compleja. El principal inconveniente es que el modelo físico no puede operar correctamente. Además, el proyecto del rediseño todavía se encuentra en desarrollo para la fecha de la reunión y se finaliza el 31 de agosto del 2023. Esto quiere decir que este proyecto coincide en el desarrollo del rediseño y no se cuenta con un modelo físico disponible para la validación. El cliente menciona que es posible utilizar un modelo de robot alternativo siempre y cuando sea de tipo manipulador.

El estado actual de la línea de investigación en robótica cognitiva de LIANA es comenzar el desarrollo de capacidades de operación básicas. Se necesita realizar rutinas de movimiento que utilicen las articulaciones del robot. El sistema cuenta con dos brazos idénticos y se busca que no ocurran colisiones. Esto implica que las acciones atómicas deben implementar prevención de colisiones dentro de su lógica. Otra consideración importante es que la tecnología usada para el ambiente de robótica cognitiva está obsoleto y se debe actualizar.

Debido a que el modelo físico del robot se debe reconstruir en base al rediseño [4] otro enfoque del laboratorio en esta área es obtener fondos para la compra de los materiales necesarios. La principal necesidad que justifica la búsqueda de fondos es incrementar

el impacto y aplicabilidad de los resultados de la investigación y desarrollo de acciones atómicas de robótica cognitiva.

En cuanto a la segunda y tercer pregunta, el cliente indica que una acción atómica se considera como una función indivisible que un robot realiza. Por ejemplo, mover la tenaza del robot a una posición específica y manipular objetos. Esto involucra movimientos coordinados de las articulaciones del robot. Además se destaca que las acciones atómicas deben tener un alto nivel de repetibilidad y ser capaces de ser parametrizadas.

Para la cuarta pregunta, el cliente responde que algunos parámetros posibles de variación son: las dimensiones de los objetos que se deben manipular, las coordenadas a las cuales se puede viajar, el espacio que haya entre los brazos robóticos y la cantidad de grados que debe girar la articulación. El espacio de trabajo del sistema de robótica cognitiva del laboratorio debe ser muy controlado y se deben evitar perturbaciones externas.

Al realizar la quinta pregunta al cliente se averigua que si se necesita implementar capacidades de visión artificial en el sistema. No obstante, se menciona que esta implementación se sale del alcance de las acciones atómicas y no se puede contemplar en este proyecto.

Por último, se establece que el ambiente de desarrollo de las acciones debe ser el mismo que el de la computadora del laboratorio que controlará a *OSCAR*. El ambiente es Linux y se desea aprovechar el kit de desarrollo de software que se llama Robot Operating System *(ROS)*. Una observación importante por parte del cliente es que la versión de *ROS* que se usa para *OSCAR* quedará obsoleta en 2025. El laboratorio tiene interés en generar conocimiento sobre la nueva versión *ROS* 2 para actualizar su base de datos.

3.2.2. Apuntes principales de la entrevista con el cliente

A continuación, se resumen las principales observaciones ingenieriles de la situación planteada:

1. No hay un modelo físico disponible para realizar pruebas de las acciones atómicas.
2. Se necesitan fondos para realizar una reconstrucción al modelo físico de *OSCAR*.
3. Reconstruir el modelo físico es una tarea cuyo alcance puede que se deba excluir de este proyecto debido a la cantidad de procesos involucrados.
4. Se deberán considerar posibilidades alternativas para efectuar la validación de los resultados de las acciones atómicas por desarrollar.
5. Una acción atómica es una implementación de lógica de software para controlar de forma confiable y coordinada las articulaciones de un robot para alcanzar un objetivo específico.
6. El cliente deja a criterio del diseñador las acciones que se van a implementar. Se solicita que se escojan movimientos simples pero que se puedan combinar entre si.

7. Este proyecto tiene riesgos económicos implicados que pueden sustentar un análisis económico.

8. El cliente muestra un sesgo para la tecnología requerida para la solución porque establece de forma clara que debe ser en ambiente Linux y con el kit de desarrollo de software *ROS*.

9. El cliente está abierto a utilizar un método de validación alternativo al modelo físico *OSCAR* con tal de actualizar la arquitectura cognitiva a una versión más moderna de *ROS* 2.

10. El cliente establece que desea implementar capacidades operativas en un robot de tipo **manipulador** como lo es *OSCAR*. Se desea investigar como funcionan las bibliotecas de desarrollo de software de *ROS* 2 y implementar una arquitectura compatible para que una vez se tenga acceso a un modelo físico de *OSCAR* se pueda controlar de forma sencilla.

3.2.3. Determinación de necesidades

Luego de analizar de forma detenida todo lo que el cliente Esteban Arias Mendez menciona en la reunión se plantea la tabla 3.1. En esta tabla se puede observar las necesidades de este proyecto. También se indica la pregunta al cliente que origina la necesidad para sustentar su validez. Se utiliza *SD* para indicar Sistema Diseñado y ahorrar espacio en la tabla.

Tabla 3.1: Necesidades que el cliente necesita resolver

No.	Necesidad	Pregunta originadora
1	El SD controla el robot para moverse a una posición específica	2, 3, 4
2	El SD permite rotar una cantidad variable de grados la articulación indicada	3
3	Se asegura que el SD evite colisiones en el manipulador	2
4	El ambiente de desarrollo del SD es *ROS*	6
5	Se resuelve el riesgo económico del SD o plantear un plan de remedio para validar su funcionamiento	1

Para redactar las necesidades se siguen los lineamientos que se establecen en la metodología de diseño [30]. Las reglas más importantes que fueron consideradas son:

1. La necesidad se redacta en términos de la funcionalidad deseada del producto y no en términos de la forma en la que se podría resolver. De esta forma se evitan sesgos.

2. Sólo se puede usar la información recopilada en la entrevista con el cliente y no se puede afirmar detalles que no se hablaron con el cliente.

3. Para redactar una necesidad se debe evitar las palabras debe y debería.
4. Las necesidades deben ser afirmaciones siempre y cuando sea posible. Las negaciones no son recomendables usar.
5. Las necesidades se deben enunciar como los atributos deseados del producto final.

3.2.4. Análisis de problemática Ishikawa

Un diagrama de Ishikawa es una herramienta que permite analizar las causas de una problemática. Se usa como referencia la técnica de las 6M [31] para analizar las distintas causas principales del problema. Las espinas analizadas son: Mano de obra, maquinaria, materiales, método, medio ambiente y medida. En la Fig. 3.2 se puede observar el diagrama de Ishikawa para este proyecto. En total se establecen 6 subcausas a la problemática principal que se debe abarcar.

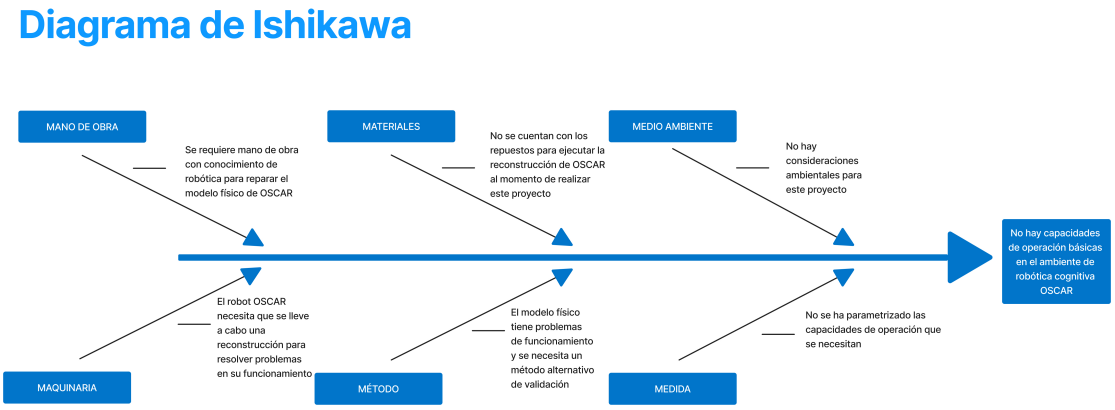


Figura 3.2: Diagrama de Ishikawa para analizar la problemática. Elaboración propia: Figma

3.2.5. Asignación de importancia de las necesidades

Se comunica al cliente un formulario para asignar un nivel de importancia a las necesidades del proyecto. En la tabla 3.1 se resumen los niveles de importancia para el proyecto.

Tabla 3.2: Niveles de importancia y significado

No.	Importancia	Interpretación
1	Función indeseable	No se aceptaría un producto con esta función
2	Función no importante	Su presencia es indiferente en el resultado
3	Función buena	No es necesaria su implementación
4	Función necesaria	Se consideraría un producto que no la tuviera
5	Función crítica	Debe estar presente en la solución

Luego de establecer una escala de medición de importancia se toman las necesidades de la tabla 3.1 y junto con el cliente se asignan los niveles correspondientes. De esta forma se genera la tabla 3.3. Se observa que las acciones atómicas de movimiento a una coordenada y rotación de articulaciones son las más importantes. Las acciones de tomar objetos y rotar objeto tienen una importancia menor. El cliente considera más importante tener disponible acciones de movimiento que permitan analizar el rango de movimiento del ambiente de robótica.

Tabla 3.3: Necesidades que el cliente necesita resolver

No.	Necesidad	Importancia
1	El SD controla el robot para moverse a una posición específica	5
2	El SD permite rotar una cantidad variable de grados la articulación indicada	4
3	Se asegura que el SD evite colisiones en el manipulador	4
4	El ambiente de desarrollo del SD es *ROS*	5
5	Se resuelve el riesgo económico del SD o plantear un plan de remedio para validar su funcionamiento	5

3.2.6. Jerarquización de necesidades

La jerarquización de necesidades corresponde al agrupamiento de las necesidades identificadas en una necesidad general que se debe redactar. A continuación se muestra el listado jerarquizado de las necesidades establecidas para este proyecto.

1. El SD implementa capacidades de operación básicas en un ambiente de robótica cognitiva
    1.1. El SD controla el robot para moverse a una posición específica
    1.2. El SD permite rotar una cantidad variable de grados la articulación indicada
2. El SD tiene un ambiente de pruebas controlado
    2.1. Se asegura que el SD evite colisiones en el manipulador
    2.2. El ambiente de desarrollo del SD es *ROS*
3. El costo del SD se ajusta al presupuesto
    3.1. Se resuelve el riesgo económico del SD o plantear un plan de remedio para validar su funcionamiento

Se destaca que en total hay tres categorías de necesidades. La mayoría de necesidades son relacionadas a capacidades de operación básicas del ambiente de robótica. También se consideran necesidades del ambiente de pruebas y del costo de la solución.

3.2.7. Categorización de necesidades

A continuación, en la tabla 3.4 se resume la categorización de necesidades. La principal observación es que todas las necesidades identificadas son indispensables de resolver. No hubo ninguna necesidad que fuera rechazada por el cliente.

Tabla 3.4: Necesidades que el cliente necesita resolver

No.	Necesidad	Categoría
1	El SD controla el robot para moverse a una posición específica	Indispensable
2	El SD permite rotar una cantidad variable de grados la articulación indicada	Indispensable
3	Se asegura que el SD evite colisiones en el manipulador	Indispensable
4	El SD puede agarrar un objeto de prueba cilíndrico	Indispensable
5	El SD es capaz de rotar un objeto de prueba cilíndrico	Indispensable

3.2.8. Establecimiento de especificaciones

Ahora que ya se conocen de forma clara las necesidades del cliente y se han asignado niveles de importancia se debe proceder a establecer especificaciones. Según la metodología de diseño se debe esperar un comportamiento diagonal en esta tabla, lo cual significa que cada necesidad tiene al menos una métrica con cual medirse [30]. Es importante mencionar que en este proceso se establecen las métricas pero todavía no se conocen los valores objetivo.

Tabla 3.5: Matriz de Necesidades y Métricas de la solución

No.	Necesidad	Métrica 1 Precisión de rotación	Métrica 2 Porcentaje de colisiones	Métrica 3 Plataforma de desarrollo	Métrica 4 Costo Unitario
1	El SD controla el robot para moverse a una posición específica	X			
2	El SD permite rotar una cantidad variable de grados la articulación indicada	X			
3	Se asegura que el SD evite colisiones en el manipulador		X		
4	El ambiente de desarrollo del SD es ROS			X	
5	Se resuelve el riesgo económico del SD o plantear un plan de remedio para validar su funcionamiento				X

3.2.9. Análisis para justificar selección de métricas

En total se escogen 4 métricas para medir el cumplimiento de las necesidades del cliente. Ahora, es importante destacar las consideraciones principales que justifican la selección. En la tabla 3.6 se encuentran resumidas las especificaciones con sus valores objetivo correspondientes.

Precisión de rotación

Los robots son sistemas mecánicos que tienen inexactitudes inherentes. En robótica, la precisión de los movimientos depende de la resolución espacial, la exactitud y la repetibilidad. La resolución espacial es el incremento más pequeño de movimiento que el robot puede realizar. La exactitud del movimiento es la capacidad de situar el extremo de la tenaza en un punto específico del espacio de trabajo. La repetibilidad corresponde a que el robot pueda moverse de forma confiable en el espacio de trabajo cuantas veces sean necesarias [32]. En aplicaciones de robótica la unidad de distancia de las aplicaciones es el metro *(m)*.

Las articulaciones del ambiente de robótica en manipuladores permiten rotación sobre si mismas similar al movimiento de una bisagra de una puerta [33]. En este caso, las unidades de medición serán grados. Se plantea el análisis de la precisión de rotación. En

base a la diferencia que haya entre la rotación indicada y la rotación realizada se puede establecer un umbral que permita la medición del comportamiento del robot. Además, si se repiten pruebas y se obtienen diferencias similares en todas pruebas se puede evaluar la repetibilidad de los movimientos del robot. Los valores objetivo se escogen de forma que habrá una diferencia máxima de un grado entre la posición indicada y la alcanzada.

Porcentaje de colisiones

El porcentaje de colisiones es importante medirlo para proteger el robot de daños a si mismo y a los operarios [33]. Se escoge medir las colisiones como porcentaje debido a que se necesita demostrar que las acciones atómicas son repetibles. La validación de este tipo de implementación requiere la realización de un número significativo de repeticiones. El porcentaje representará la cantidad de veces que ocurren colisiones durante el proceso de prueba. Idealmente, se alcanza un porcentaje de colisiones de cero. No obstante, un valor marginal de cinco sigue siendo apropiado para demostrar las bajas ocurrencias de colisiones.

Plataforma de desarrollo

La elección de *ROS (Robot Operating System)* como plataforma de desarrollo es una necesidad importante que presenta sesgo en cuanto a la tecnología utilizada para la solución. *ROS* proporciona un marco de trabajo ampliamente utilizado en robótica y es compatible con una variedad de robots y sensores. Para medir el cumplimiento de esta necesidad, se puede evaluar si el sistema de acciones atómicas está diseñado y funcionando correctamente en el entorno *ROS*. Esto implica verificar si se pueden utilizar las herramientas y bibliotecas de *ROS* para desarrollar, ejecutar y depurar el software del robot. Es importante destacar que hay múltiples versiones de *ROS* pero se debe optar por la más reciente para asegurar que el proyecto tenga soporte a largo plazo [34]. Esta métrica es binaria y el único valor objetivo aceptable es *ROS*.

Costo unitario

El costo de desarrollar acciones atómicas es una preocupación importante, especialmente si se están considerando riesgos económicos. Para evaluar esta métrica, es necesario realizar un análisis detallado de los costos asociados al desarrollo y despliegue del sistema. Esto incluye los costos de hardware, software, mano de obra, materiales y cualquier otro recurso necesario. Además, se debe comparar el costo total con el presupuesto disponible para determinar si se ajusta a las restricciones económicas del proyecto. El método de validación para este proyecto depende del costo unitario de la solución. Se medirá en dólares. El costo máximo de este proyecto es de dos mil dólares.

Tabla 3.6: Lista de especificaciones y valores objetivo

No.	Métrica	Unidad	Imp.	Valor Marginal	Valor Ideal
1	Precisión de rotación	°	3	<1°	<0.5°
2	Porcentaje de prevención de colisiones	%	5	>95	100
3	Plataforma de desarrollo	Binaria	5	ROS	ROS
4	Costo Unitario	USD	5	<2000	<1500

3.3. Análisis económico; Estudio de factibilidad de validación

En este proyecto es importante realizar un análisis económico preliminar al desarrollo de la solución debido a que *OSCAR* necesita ser reconstruido según lo indica el cliente en la entrevista. Esto tiene implicaciones directas en el método de validación que se puede usar para las acciones atómicas solicitadas por el cliente. La economía de un proyecto depende de muchos recursos. El tiempo, dinero y materiales son los principales recursos involucrados en este proyecto. El ambiente de desarrollo es un laboratorio y cuenta con recursos limitados. En la tabla 3.7 se observa que el costo de los componentes para la reconstrucción son $919,88.

Tabla 3.7: Componentes para reconstrucción de OSCAR

No.	Componente	Cantidad	Precio Unitario	Precio total
1	Rodamiento 70mm 16014 NTN	4	$49.24	$196.96
2	Paquete de tornillos y tuercas M3 (30, 40, 45, 50)mm	1	$44.67	$44.67
3	Paquete de tornillos y tuercas M3 (50, 60, 70, 80)mm	1	$39.30	$39.30
4	Rodamiento Axial 5mm	1	$34.88	$34.88
5	TB6600 4A Stepper Motor Driver	12	$16.89	$202.68
6	Paquete de conectores coupler	1	$35.69	$35.69
7	Faja GT2 10M 6mm	1	$46.04	$46.04
8	Barra de acero de longitud 200mm, diámetro 500mm	1	$30.86	$30.86
9	Filamento PLA Strong	8	$36.10	$288.80
			Total	$919.88

3.3.1. Desarrollo de estrategias de recaudación de fondos para reconstrucción

Preferiblemente, las acciones atómicas por diseñar se desean validar por medio del uso del modelo físico de *OSCAR*. Al momento de iniciar este proyecto el 3 de Agosto, el rediseño del robot se encuentra en proceso [4]. No hay un modelo físico disponible para realizar validaciones de forma inmediata. Debido a esta razón, se invierten 8 semanas del desarrollo de este proyecto para presentar una propuesta de inversión a la Vicerrectoría de Investigación y Extensión del Tecnológico de Costa Rica. En la tabla 3.8 se muestra que en total para oficializar la propuesta con la vicerrectoría se deben cumplir con 10 entregables.

Tabla 3.8: Lista de entregables para propuesta de investigación

No.	Descripción
1	Planteamiento general del proyecto
2	Marco Teórico
3	Justificación de la investigación
4	Planteamiento de objetivos
5	Metodología
6	Plan de acción
7	Cronograma de acciones
8	Plan de divulgación de tecnología
9	Presupuesto

Cada entregable requiere de documentación detallada que describe la forma en la cual se invertirán los fondos. Para cada entregable se invierte tiempo en ir a la vicerrectoría y obtener realimentación para presentar el proyecto según el formato deseado por esta unidad. En general, el desarrollo de la estrategia de recaudación de fondos es un proceso largo en el cual se aprende la forma de operar de esta unidad del Instituto Tecnológico de Costa Rica.

Se acude a este departamento debido a que se anuncia la asignación de fondos a proyectos estudiantiles. La fecha en la cuál se oficializa la propuesta de financiamiento es el 1 de septiembre del 2023. Inicialmente el proyecto se aprueba para ser financiado el 12 de septiembre del 2023. Sin embargo, debido a que los componentes sólo pueden ser adquiridos internacionalmente se rechaza el fondo de inversión el 14 de septiembre del 2023 porque el calendario de compra de la vicerrectoría no coincide con el tiempo hábil para este proyecto de graduación.

3.3.2. Conclusión de análisis de factibilidad para selección del método de validación

La consecuencia principal es que se invierten 8 semanas de las 16 disponibles para la realización del proyecto en obtener presupuesto para la validación. Debido a que se rechaza el presupuesto, la estrategia de validación se debe cambiar a una alternativa. En este caso, la alternativa disponible es realizar una simulación del ambiente de robótica reconstruido. Además, se solicita una prórroga para efectuar el estudio de los ambientes de robótica simulados disponibles.

3.3.3. Plan de remedio

Ahora que se ha establecido que no es factible validar este proyecto utilizando el modelo físico de *OSCAR* se debe establecer la alternativa de validación para el proyecto. Debido a que se invierte una gran cantidad de tiempo en la búsqueda de fondos se debe establecer un plan de remedio que permita ajustarse al cronograma del proyecto incluyendo el periodo de prórroga.

El plan de remedio planteado consiste en utilizar un ambiente de simulación para usar las acciones atómicas desarrolladas. Un ambiente de simulación es una alternativa válida debido a que se puede visualizar el robot moviéndose en un espacio de trabajo. Además, se plantea usar otro robot de tipo manipulador más sencillo para validar las acciones atómicas desarrolladas. Por último, se plantea que el enfoque principal de este proyecto sea la investigación de tecnologías de control modernas para actualizar la base de datos de conocimiento del laboratorio que quedará obsoleta en 2025.

Dicho plan se explica al cliente con el fin de obtener su aprobación. El cliente acepta el plan de remedio. Se establece que ahora el enfoque del proyecto es generar aprendizaje en el área de software para robótica cognitiva.

3.4. Generación de conceptos

En la Fig. 3.3 se observa un diagrama de alto nivel que describe la solución de este proyecto. La entrada son señales de información de software asociadas a las acciones atómicas. Las salidas corresponden a señales de control de hardware. De esta forma se puede plantear que la implementación en *ROS* se encargará de controlar el robot manipulador.

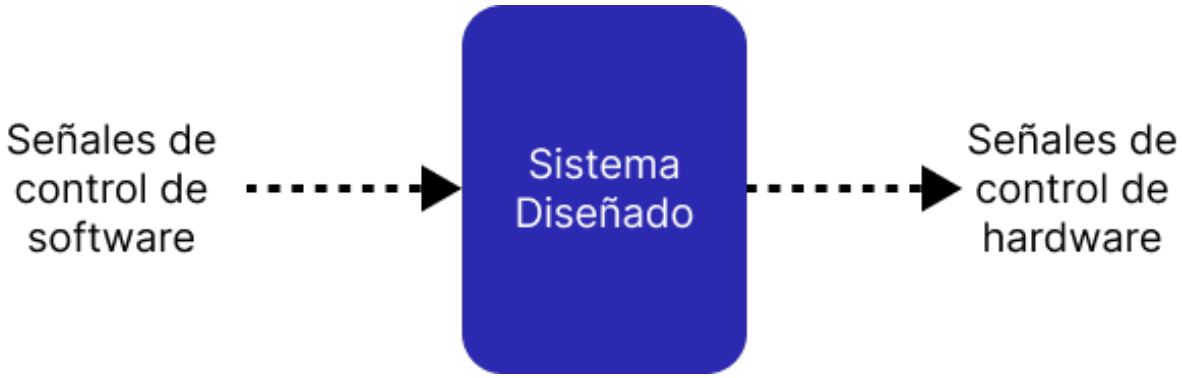


Figura 3.3: Diagrama general de la solución. Elaboración propia: Figma

Para continuar abstrayendo el problema se procede a realizar una descomposición funcional del diagrama previo. Por medio de esta descomposición se logra dividir el problema en subsistemas sin tener que definir un método de solución. De esta forma se logra una visión panorámica de los distintos módulos a desarrollar para alcanzar la solución. En la Fig. 3.4 se puede observar que se identifican cinco subsistemas que se necesitan analizar. Se destaca que este sistema sólo cuenta con señales de información como entradas y salidas. Es una integración enfocada en software que permita controlar de forma confiable y repetible el ambiente de robótica.

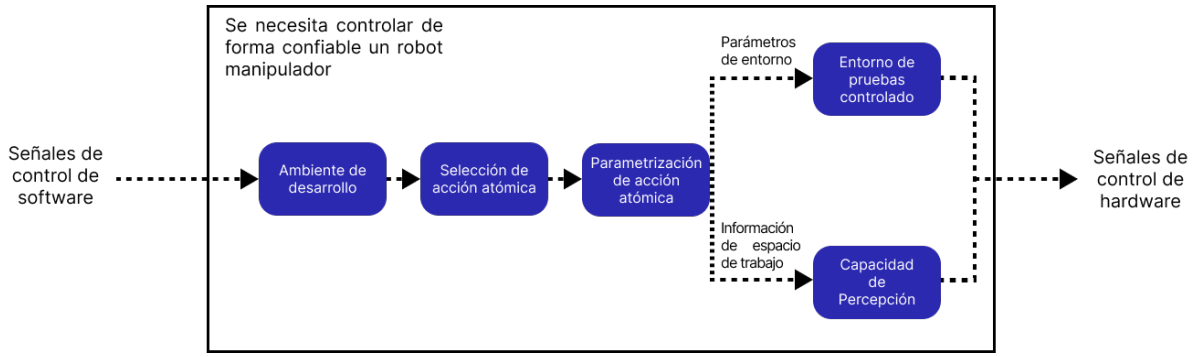


Figura 3.4: Descomposición funcional de la implementación en subsistemas. Elaboración propia: Figma

3.5. Búsqueda externa e interna

Para fundamentar cada uno de los subsistemas planteados se investigan tecnologías que puedan ser aplicadas como solución. De esta forma se genera un abanico de soluciones al problema. Luego, con la información recopilada se preparan conceptos que serán comparados entre si para determinar la solución que mejor se ajuste a las necesidades del cliente.

3.5.1. Búsqueda externa

Ambiente de Desarrollo

Es importante destacar que el cliente establece que las acciones atómicas a diseñar deben ser implementadas en el kit de desarrollo de software *ROS* [34]. La principal razón de la elección es que la línea de investigación de robótica cognitiva se ha diseñado para este ambiente. Todo el hardware es compatible con la plataforma lo cual permite implementarlo con la amplia biblioteca de funciones disponibles. Otro beneficio es que existe una comunidad activa que ha generado abundante documentación. La programación es basada en Python y C++.

Existen otros ambientes de desarrollo como PyRobot. Este es otro kit de desarrollo de software de código abierto que permite crear aplicaciones de robótica. Es desarrollado por Facebook AI Research [35]. Tiene una interfaz unificada en Python para establecer un un proceso de programación y control de robots simple. Este ambiente de desarrollo es modular y ofrece soporte para una gran variedad de robots.

Otro ejemplo de ambiente de desarrollo es Optimal Reciprocal Collision Avoidance *ORCA*. Este es un ambiente de desarrollo en C++. Su principal objetivo de uso es para sistemas de robótica que requieren navegación en entornos donde pueden ocurrir colisiones. Se destaca que es un kit de desarrollo de código abierto que permite la simulación de entornos de dos y tres dimensiones. Un factor importante es que la aplicación desarrollada en este proyecto no necesita capacidades de navegación.

En conclusión, existen múltiples posibles ambientes de desarrollo para aplicaciones de robótica. Lo más importante a considerar para la selección del ambiente son los requisitos específicos del sistema robótico que se debe implementar. Para esta solución, se considera una necesidad utilizar ROS y como consecuencia se excluyen las otras opciones encontradas en etapas posteriores.

Selección de acción atómica

La selección de acción atómica corresponde a la forma en la cual el operario utiliza el sistema diseñado para ingresar la rutina que el sistema *OSCAR* ejecutará. Es decir, es el centro de control donde se especifican los parámetros para las acciones atómicas. A continuación se desglosan algunas de las tecnologías disponibles para controlar sistemas robóticos.

Un PLC es un controlador lógico programable que es compatible con robots y es ampliamente utilizado para aplicaciones industriales. Uno de los beneficios de utilizar un PLC para controlar el sistema robótico es que se puede integrar fácilmente con otra maquinaria compatible. Otra ventaja es que la programación y mantenimiento del robot se simplifica al utilizar PLC porque se administra a través del software. No obstante, se debe considerar desde la compra o diseño que el robot debe ser compatible con este tipo de controlador.

Además, los PLC son usados en aplicaciones de nivel industrial [36].

Otra forma de considerar el control de sistemas robóticos es por medio de una computadora. Al investigar se determina que es posible el desarrollo de aplicaciones compatibles con múltiples sistemas operativos y amplias opciones de hardware que se puede usar. La ventaja de utilizar una computadora como centro de control es que el coste de implementación es bajo. Las aplicaciones controladas por computadora pueden ser de bajo nivel o hasta de nivel industrial lo cual refleja su flexibilidad. Por último, existen muchos kit de desarrollo de software que permiten implementar sistemas robóticos complejos como *ROS* [34] o *Glowbuzzer* [37]. Ambos entornos son programables y habilitan integraciones IoT.

Existen otras formas para interactuar con sistemas robóticos como lo son botoneras, palancas y controles manuales [38]. Estas tecnologías han estado disponibles desde antes que surgieran los sistemas de robótica. Ofrecen una forma de interacción confiable. No obstante, pueden ocupar mucho espacio y ser poco flexibles.

Parametrización de acción atómica

Para parametrizar acciones atómicas se solicitan los valores al operario. Se deben limitar las entradas a que se encuentren dentro de los valores objetivo establecidos en la tabla 3.6. Este proceso también se considera parte de la interacción humano máquina. Según el cliente es necesario configurar la acciones atómicas antes de que sean ejecutadas por el robot.

Una forma de solicitar al usuario de forma sencilla los parámetros de las acciones es por medio de un programa de computadora que recopile los datos. Dicha interfaz puede ser desarrollada en Python [39] o C++ [40]. Ambas plataformas cuentan con amplias opciones de interfaz gráfica. Además el ambiente de desarrollo *ROS* [34] es compatible con ambos lenguajes.

Entorno de pruebas controlado

Para incrementar la confiabilidad del desempeño de *OSCAR* se debe establecer un ambiente de pruebas controlado para disminuir la influencia de factores externos durante la realización de pruebas. En este caso, debido a que no es factible utilizar el modelo físico se puede usar tecnologías de simulación de robótica que permitan replicar el modelo reconstruido. Un opción de desarrollo es *Gazebo* [41]. En esta plataforma se pueden configurar simulaciones que consideran aspectos como:

- Modelos robóticos con sensores
- Compatibilidad con *ROS*
- Visualización 3D del ambiente
- Motor de física avanzado

Estas características hacen de esta tecnología una alternativa válida al modelo físico. Además se puede recrear el modelo reconstruido en este ambiente y de esta forma validar su correcto funcionamiento al ejecutar las acciones atómicas. Por último, la compatibilidad con *ROS* hace que sea una opción viable.

Otras opciones de simulación son *RoboDK* [42]. Este sistema es similar en capacidades que *Gazebo*. La principal limitante es que su uso requiere de una licencia.

Otra referencia de tecnología de simulación es *MoveIt* [43]. Es una plataforma de código abierto que permite planificación de trayectorias para sistemas de robótica. Además permite la percepción 3D, manipulación y detección de colisiones. Es compatible con *ROS*.

Por último, otra opción de simulación disponible en el mercado es *Delmia* [44]. Esta opción es desarrollada por *Dassault*. Esta empresa también desarrolla SolidWorks, el software de CAD usado para el diseño de *OSCAR*. Se debe destacar que no cuenta con compatibilidad con *ROS*.

Capacidad de percepción

La capacidad de percepción de un humano implica detectar señales de sonido, tacto o imagen del ambiente. En el caso de un robot, la capacidad de percepción de un robot se puede emular por medio del uso de sensores [45]. Al detectar una señal con un sensor se pueden tomar decisiones de control sobre el robot manipulador y de esta forma operarlo.

Una forma de interpretar la tarea a realizar es por medio del uso de comandos de voz. Esta es una forma intuitiva de comunicarse con el robot. No obstante, lograr que un programa de computadora sea capaz de interpretar una instrucción dicha en un micrófono no es tarea sencilla. Para este problema, existen tecnologías con modelos de lenguaje como Alexa [46]. Este asistente de voz ya cuenta con un robusto sistema de interpretación de lenguaje y además ofrece sus servicios por medio de *API* [47] lo cual permite integrar su sistema con *ROS* 2. De esta forma, se puede usar un micrófono para que el robot perciba una instrucción objetivo.

El robot *OSCAR* cuenta con capacidad de visión artificial por medio de una cámara, esta es otra forma de percibir cambios en el ambiente. Esto significa que se debe detectar distintos objetos en el campo de visión. Para este propósito existe *OpenCV* [48]. Esta es una librería de código abierto basada en Python. Esto significa que es compatible con *ROS*, lo cual hace esta tecnología viable.

Otra opción en el mercado es *Scikit* [49]. Esta opción también es de código abierto y se desarrolla en Python. Cuenta con compatibilidad con *ROS*. Finalmente, otra opción viable es *Pillow* porque es de código abierto, basada en Python y compatible con *ROS* [50].

3.5.2. Búsqueda interna

Por medio del uso de árboles de organización conceptual se puede observar de forma clara todas las tecnologías referenciadas en la etapa previa. En la Figs. 3.5, 3.6, 3.7, 3.8 y 3.9 se encuentran los árboles. Como principal observación de los árboles preparados se destaca que no se separan en subcategorías ya que todas las opciones listadas dependen de software.

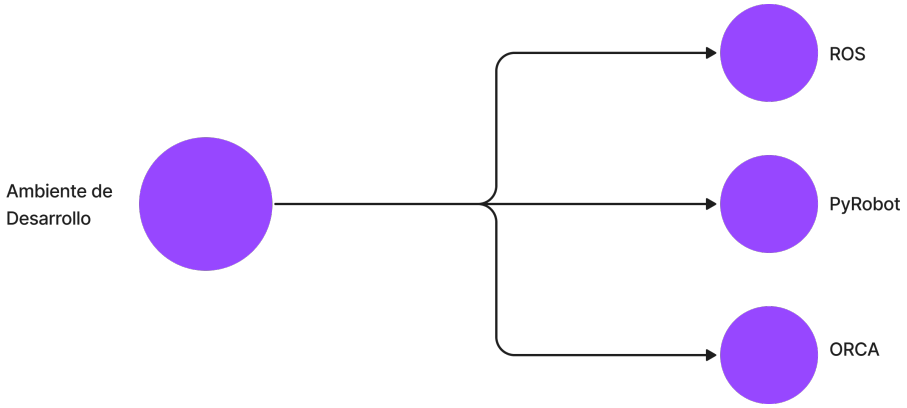


Figura 3.5: Árbol para ambiente de desarrollo. Elaboración propia: Figma

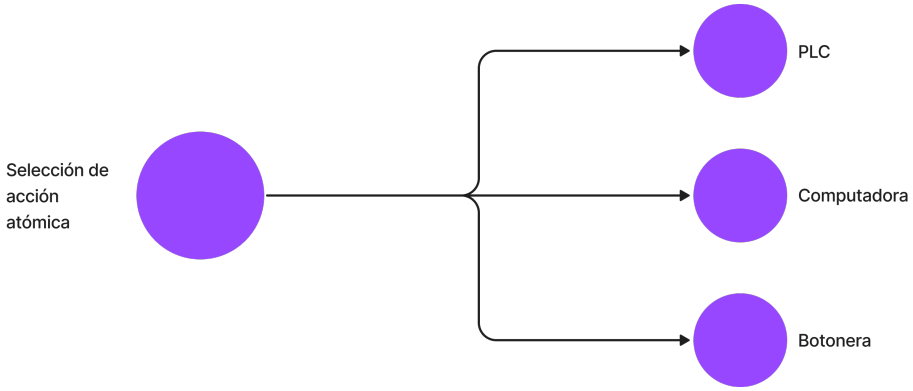


Figura 3.6: Árbol para selección de acción atómica. Elaboración propia: Figma

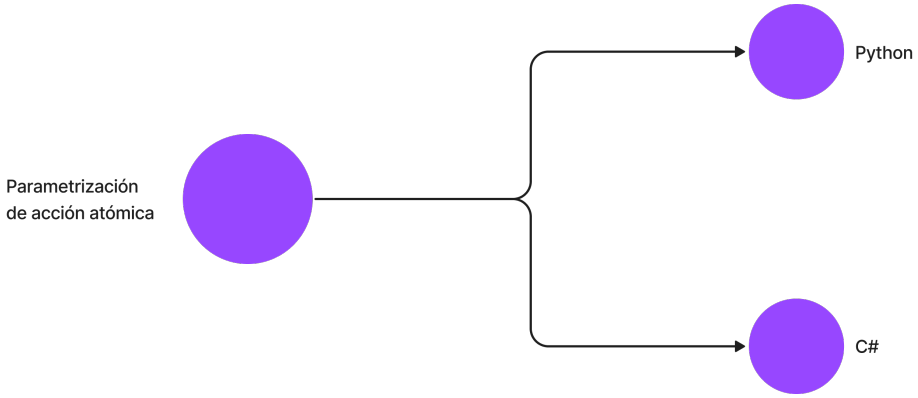


Figura 3.7: Árbol para parametrización de acción atómica. Elaboración propia: Figma

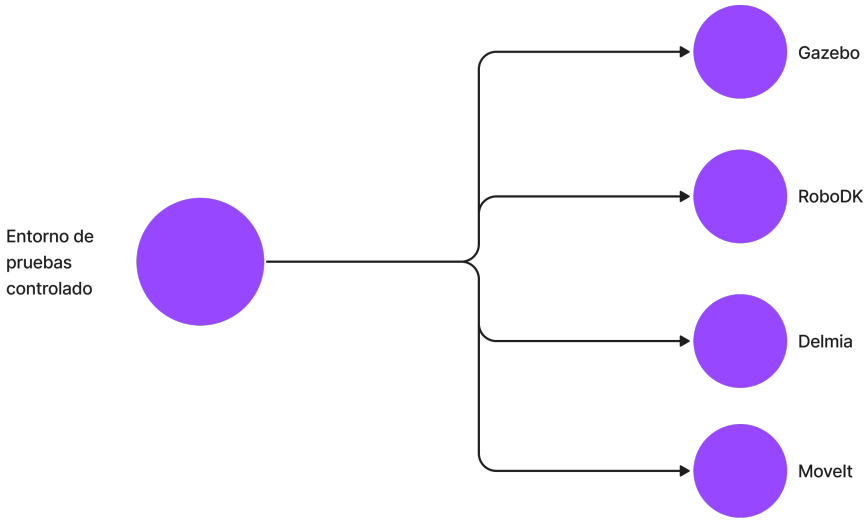


Figura 3.8: Árbol para entorno de pruebas controlado. Elaboración propia: Figma

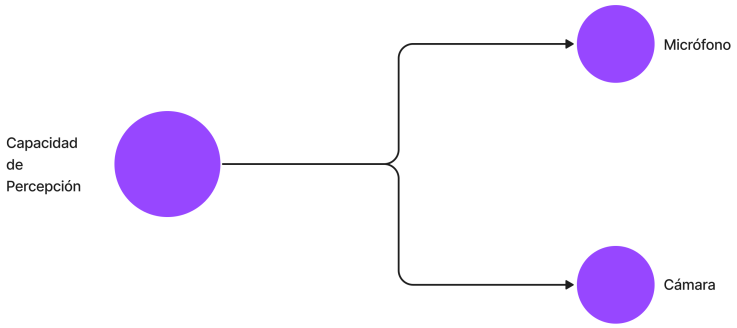


Figura 3.9: Árbol para capacidad de percepción. Elaboración propia: Figma

3.6. Generación de conceptos de solución

Ahora que ya se conocen tecnologías aplicables para la solución y se han organizado en árboles de categorías de conceptos se procede a generar distintas permutaciones de conceptos. En la Fig. 3.10 está la plantilla con la que se generan conceptos de solución.

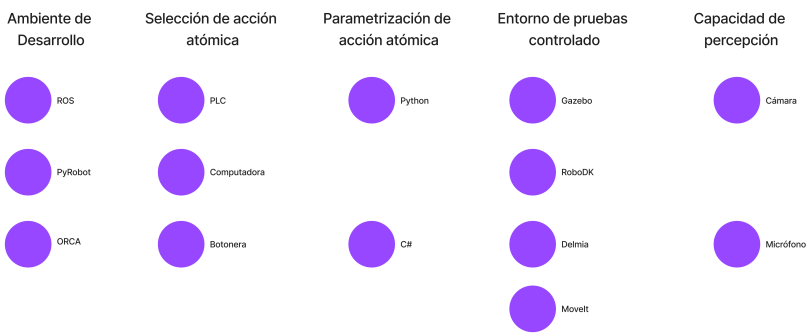


Figura 3.10: Plantilla para generación de conceptos. Elaboración propia: Figma

El primer concepto generado se observa en la Fig. 3.11. Tiene las siguientes caracterísiticas:

- Ambiente de desarrollo en ROS.
- Se seleccionarán las acciones atómicas con una computadora.
- Se parametriza las acciones atómicas en Python.
- Se simulará el entorno de pruebas controlado en Gazebo.
- Se utiliza un micrófono como sensor para integrar la capacidad de percepción al robot.

Figura 3.11: Primer concepto de solución. Elaboración propia: Figma

El segundo concepto corresponde a la Fig. 3.12 y sus características se listan a continuación:

- Ambiente de desarrollo en *ROS*.
- Se seleccionarán las acciones atómicas con un PLC.
- Se parametrizarán las acciones atómicas en Python.
- Se simulará el entorno de pruebas controlado en RoboDK.
- Se usa una cámara para percibir el espacio de trabajo.

Figura 3.12: Segundo concepto de solución. Elaboración propia: Figma

El tercer concepto corresponde a la Fig. 3.13 y sus características se listan a continuación:

- Ambiente de desarrollo en *ROS*.
- Se seleccionarán las acciones atómicas con una botonera.
- Se parametrizarán las acciones atómicas en C.
- Se simulará el entorno de pruebas controlado en MoveIt.
- Se percibe el ambiente con un micrófono.

Figura 3.13: Tercer concepto de solución. Elaboración propia: Figma

El cuarto concepto corresponde a la Fig. 3.14 y sus características se listan a continuación:

- Ambiente de desarrollo en *ROS*.
- Se seleccionarán las acciones atómicas con una computadora.
- Se parametrizarán las acciones atómicas en C.
- Se simulará el entorno de pruebas controlado en Delmia.
- Se usa una cámara como sensor.

Figura 3.14: Cuarto concepto de solución. Elaboración propia: Figma

3.6.1. Selección de concepto

En esta sección se comparan los conceptos generados usando criterios de diseño que permiten cuantificar la mejor solución. Los criterios de diseño que se usan son:

- Compatibilidad con *ROS*: Es un requisito del cliente usar este kit de desarrollo de software para controlar el robot.
- Costo: Es un criterio debido a que el costo limita las tecnologías que se pueden utilizar como solución.
- Apertura de integraciones: Es importante que la tecnología de simulación permita integrarse de forma sencilla con la tecnología de control.
- Innovación: Se considera si el laboratorio ya cuenta con conocimiento de este tipo de tecnología. Este criterio aplica específicamente al sensor utilizado para dar capacidades perceptivas al robot.
- Flexibilidad de control: Este criterio evalúa la capacidad de control remoto que tiene el concepto.

En la tabla 3.9 se resume el proceso de comparación. Se escoge el concepto 2 como la referencia debido a que es común controlar sistemas de robótica con PLC como centro de control en escenarios industriales. Se marca con + los criterios que son mejores respecto a la referencia y con − los que son peores. El concepto ganador es el 1 debido a las siguientes razones:

- Tiene un menor costo de implementación que la referencia.
- La tecnología de simulación Gazebo es completamente compatible con la tecnología de control *ROS*.
- El sistema tiene mayor flexibilidad de control que la referencia debido a que se usa una computadora como estación de control.
- Ofrece mayor innovación porque el laboratorio no cuenta con investigación de implementación de sistemas de reconocimiento de lenguaje como *Alexa*. Cuando se describe *OSCAR* [2] se implementa con una cámara por lo que ya se cuenta con conocimiento de este sensor. Se considera valioso usar un micrófono como sensor para ampliar el conocimiento de la línea de investigación.
- Hace uso de tecnologías abiertas, lo cual facilita su replicación y soporte a largo plazo.

El concepto 3 se descarta debido que utiliza C++ para controlar el robot pero la simulación es basada en Python lo cual implica menor compatibilidad entre tecnologías. Además, debido a que se plantea controlar el sistema con una botonera hay menor flexibilidad de control debido a que este equipo no se puede usar de forma remota.

Por último, el concepto 4 cuenta con limitaciones similares a las anteriores. La principal diferencia es que debido a que el sistema de simulación se plantea por medio de Delmia el sistema cuenta con una compatibilidad con *ROS* reducida y una comunidad de soporte

exclusiva en comparación a *Gazebo*. Esto corresponde a una característica del software Delmia según lo investigado.

Tabla 3.9: Comparación de conceptos por medio de criterios de diseño

Criterio	Concepto			
	1	2 (Ref)	3	4
Compatibilidad con ROS	0	0	0	-
Costo	+	0	0	-
Apertura de integraciones	+	0	-	+
Innovación	+	0	+	0
Flexibilidad de control	+	0	-	-
Suma +	4	0	1	1
Suma 0	1	5	2	1
Suma -	0	0	2	3
Sumatoria	4	0	-1	-2
Ranking	1	2	3	4
¿Continua?	Si	Si	No	No

En esta etapa se pueden identificar conceptos que se pueden combinar para obtener una mejor solución. Sin embargo, se observa que los resultados demuestran que el concepto 1 no se puede beneficiar de alguna tecnología de los otros conceptos.

3.6.2. Evaluación de conceptos

Para finalizar la selección del concepto solución se utiliza la matriz de evaluación de conceptos propuesta en la tabla 3.10. Por medio de esta tabla se obtiene una calificación que permite escoger el concepto de forma cuantitativa. Para cada criterio se establece una importancia que refleje las necesidades del cliente.

Tabla 3.10: Matriz de evaluación de conceptos 1 y 2

Criterio	Peso	Concepto			
		2 (Ref)		1	
		Calificación	Ponderado	Calificación	Ponderado
Compatibilidad con ROS	20	3	0.6	3	0.6
Costo	40	3	1.2	5	2
Apertura de integraciones	10	3	0.3	4	0.4
Innovación	10	3	0.3	5	2
Flexibilidad de control	20	3	0.6	4	0.8
Total	100	3		5.8	
Ranking		2		1	
¿Continua?		No		Desarrollar	

Las principales observaciones son:

- El criterio de mayor peso es el costo debido a que es el recurso que más limita las tecnologías que se pueden usar como solución.
- La mayor ventaja que el concepto 1 tiene en comparación al 2 es un costo mucho menor debido a que no se cuenta con un PLC disponible en laboratorio para controlar el robot y se tendría que adquirir.
- Otra ventaja importante es en la apertura de integraciones del concepto ganador 1 debido a que se controlará el sistema con una computadora lo cual permite implementar software que se puede integrar con mayor facilidad en comparación a un PLC.
- La flexibilidad de control del concepto ganador es mayor debido a que una computadora ofrece más opciones de control remoto en comparación a un PLC que requiere de mayor esfuerzo para controlar remotamente.
- En cuanto a la capacidad de medición, ambas opciones permiten la opción de implementar sensores en el sistema pero el PLC ofrece una mayor robustez para las mediciones debido a que es para aplicaciones industriales.
- En cuanto a la innovación, se obtendrá una implementación de comandos de voz en el ambiente de robótica cognitiva.

Según los datos previamente detallados, se concluye que el concepto que mejor se ajusta a las necesidades del cliente es el 1. Es el concepto que tiene menor costo y tiene mayor flexibilidad en la integración del sistema con el robot. Además tiene la ventaja de que el ambiente de simulación es compatible con *ROS*.

3.7. Diseño de pruebas de validación

Ahora que ya se cuenta con un concepto ganador, se deben plantear pruebas que permitan validar el cumplimiento correcto de las especificaciones de la tabla 3.6.

3.7.1. Primer prueba de concepto: Simulación de movimiento de brazo

Para esta prueba se plantea la validación de las acciones atómicas:

1. Mover robot a posición de descanso
2. Mover robot a posición de trabajo
3. Mover robot a posición objetivo

El objetivo de esta prueba es corroborar que la simulación permite al usuario interactuar con el robot y enviarle instrucciones. En esta prueba, el robot tiene que alcanzar las posiciones al recibir el comando. La naturaleza de prueba es para verificar el correcto funcionamiento de las capacidades de movimiento en el ambiente de simulación. **Variable a medir:** Lista de capacidades de movimiento. En este caso, se declaran 3 capacidades atómicas de movimiento en el robot.

Cantidad de pruebas a realizar: Para verificar que el funcionamiento de las acciones atómicas se realizará una prueba de movimiento para cada pose objetivo. En este caso, basta con una prueba única porque la simulación ofrece un modelo controlado y determinista.

3.7.2. Segunda prueba de concepto: Simulación de rotación de articulaciones

Se valida la acción atómica de rotación de una articulación de los brazos robóticos utilizando el ambiente simulado. Para esto, se escoge la articulación y se indica la cantidad de grados a rotar. Se deben probar todas las articulaciones en 3 posiciones, rotación mínima, intermedia y máxima.

Variable a medir: Diferencia entre ángulo final de la articulación del robot y el ángulo objetivo.

Cantidad de pruebas a realizar: Se realizan 30 pruebas por articulación para obtener una muestra estadísticamente relevante [51]. De estas pruebas se obtendrá un promedio de la variable medida en cada prueba. Se repite este proceso para cada articulación para determinar que todas cumplan las especificaciones.

3.7.3. Tercera prueba de concepto: Simulación de prevención de colisiones

Se valida la capacidad del sistema robótico de evitar colisiones entre los brazos. Se utilizará la acción de movimiento de los brazos para plantear 5 trayectorias válidas y 5 inválidas. El experimento consiste en verificar que el sistema no realice las trayectorias inválidas y que retorne una advertencia. Para las trayectorias válidas se plantea retornar un mensaje de éxito.

Variable a medir: Porcentaje de evitación de colisiones. Cada repetición tendrá un resultado binario de colisión o éxito. El porcentaje de colisión se calcula de la siguiente forma:

$$\text{Porcentaje de prevención de colisiones} = 100 * \frac{\text{Cantidad de Colisiones}}{\text{Cantidad de pruebas realizadas}} \quad (3.1)$$

Cantidad de pruebas a realizar: En total, se van a realizar 10 pruebas. 5 pruebas son de trayectoria válida y las otras 5 son de un movimiento inválido.

3.8. Análisis económico, social y ambiental

A continuación, se analiza el consumo y generación de recursos que la realización de este proyecto tiene en los ámbitos económicos, sociales y ambientales. El principal objetivo es justificar el desarrollo de este proyecto demostrando que habrá un retorno beneficioso para el laboratorio.

3.8.1. Consideraciones sociales

Este proyecto se desarrolla en el laboratorio *LIANA*. Uno de sus principales objetivos es generar nuevo conocimiento y habilidades en la comunidad estudiantil. En este caso, el conocimiento generado es en robótica cognitiva, un área de alta relevancia para la ingeniería mecatrónica.

Desde un punto de vista social, la realización de este proyecto se justifica debido a que continúa una línea de investigación relevante y robustecerá la base de datos existente de robótica cognitiva. La robótica cognitiva es utilizada en la industria médica, alimenticia y metalúrgica debido a que reduce las posibilidades de error humano y incrementa la confiabilidad en procesos de manufactura.

3.8.2. Consideraciones ambientales

Este proyecto consta de una implementación de control para un robot cognitivo. Esta estrategia de control es principalmente en software. Además, como se demostró en el

análisis económico para la factibilidad de validación se debe simular el ambiente robótico. Esto quiere decir que el consumo de electricidad es el principal insumo para desarrollar este proyecto. Por lo tanto, se puede considerar que tiene un bajo impacto ambiental.

Sin embargo, es importante considerar que el modelo físico del robot todavía requiere ser reconstruido para que se pueda usar en pruebas de laboratorio que aprovechen las acciones atómicas. Algunos factores de importancia ambiental son los siguientes:

- La reconstrucción del robot implica utilizar impresión 3D para las piezas rediseñadas de la carcasa. El material escogido para la reconstrucción es PLA Strong. Este material plástico tiene características negativas para el ambiente ya que puede tardar más de 100 años en degradarse [52].
- La utilización de impresoras 3D tiene un consumo energético inherente que puede tardar días según la complejidad de la pieza. También se generan gases.
- Reconstruir el robot significa que las piezas reemplazadas tendrán que ser desechadas. Debido a que hay desechos electrónicos se deben procesar de forma adecuada para minimizar el impacto ambiental.

El alcance de este proyecto cubre la implementación de un robot manipulador en un ambiente simulado implementado con *ROS* 2. La reconstrucción de *OSCAR* no es parte del alcance pero su impacto ambiental es importante de destacar.

3.8.3. Consideraciones económicas

Ya se ha realizado un análisis económico para justificar la selección del método de validación. Ahora en este apartado se estudia el costo histórico de la línea de investigación y el costo de este proyecto.

Costo histórico de la línea de investigación

En total se han completado 3 proyectos. La tabla resume el costo que cada proyecto ha conllevado.

Tabla 3.11: Costo histórico en colones de la línea de investigación *OSCAR*.

No.	Nombre de proyecto	Costo
1	Diseño de entorno robótico como herramienta para el desarrollo de arquitecturas cognitivas [2]	956338
2	Implementación de un sistema robótico para la realización de experimentos avanzados de robótica [3]	1507021
3	Rediseño del Robot OSCAR Atendiendo Requerimientos de Fiabilidad de Operación en Condiciones Demandantes [4]	1715051
Total		4178410

En el proyecto 1 los principales recursos invertidos en el desarrollo del ambiente de robótica *OSCAR* son el tiempo y dinero. El proyecto 2 también implica costo de tiempo de desarrollo pero muestra la diferencia de que se invierte en materiales para la construcción del modelo físico. En el proyecto 3 también se plantea la compra de repuestos y el costo del rediseño. En total, la línea de investigación tiene un valor de aproximadamente 4 millones de colones.

Los beneficios devueltos para el laboratorio son:

- Una amplia base de datos que especifica el ambiente de robótica cognitiva *OSCAR*.
- Un modelo físico para pruebas de laboratorio que debe ser reconstruido.
- Desarrollo de habilidades en estudiantes que obtuvieron un grado académico de ingeniería mecatrónica por presentar los proyectos.

Como conclusión, el conocimiento generado en la línea de investigación es un insumo esencial para desarrollar este proyecto porque establece el tipo de robot que se necesita controlar con las acciones atómicas solicitadas.

Costo del desarrollo de acciones atómicas

En este apartado se declara el consumo de recursos de este proyecto. Una de las principales ventajas del concepto ganador diseñado es que está basado en software de uso libre. La codificación del modelo del robot y su simulación no tendrán costo. El tiempo de desarrollo de la solución representa un recurso que se debe registrar. También se debe considerar el costo del pago de la licencia del software de edición de código LaTex para digitar la documentación. Esta licencia cuesta $7 por mes. La tabla 3.12 muestra desglose de los costos:

Tabla 3.12: Componentes necesarios para desarrollar el concepto ganador

No.	Componente	Cantidad	Costo
1	Software ROS y Gazebo	1	0
2	Computadora personal	1	1500000
3	Overleaf	8 meses con pagos de 3312.5	26500
4	Computadora de laboratorio	1	500000
		Total	2026500

La depreciación corresponde a la pérdida de valor que un activo tiene a lo largo de su vida útil [53]. Esta consideración aplica para las computadoras que se utilizan para desarrollar la solución. Las ecuaciones muestran las fórmulas para calcular este factor.

$$\text{Valor de desecho} = \frac{\text{Valor inicial}}{\text{Años de vida útil}} \tag{3.2}$$

$$\text{Depreciación} = \frac{\text{Valor inicial- Valor de desecho}}{\text{Años de vida útil}} \tag{3.3}$$

Al utilizar las fórmulas anteriores para las computadoras con una vida útil de 5 años se determina que la computadora personal se deprecia 240000 colones y la del laboratorio 80000 colones. En total se contabilizan 320000 colones asociados a la depreciación del equipo usado para la solución. La inversión de tiempo de desarrollo de la solución se calcula de la siguiente forma:

- El periodo de realización del proyecto es 16 semanas del II Semestre del 2023 y 8 semanas del I Semestre del 2024. En total se contabilizan 24 semanas de trabajo.
- Por semana se invierten 30 horas de desarrollo del proyecto, investigación de fuentes bibliográficas y documentación de los resultados. Además se realizan revisiones de progreso con el cliente.
- En total se invierten 720 horas en el desarrollo de la solución. El costo de hora de asistencia estudiantil es de 600 colones por hora. El costo final es de 432000.

En la tabla 3.13 se observa que el costo total por el desarrollo de la solución es de 752000 colones. Sin embargo, el laboratorio puede ahorrarse este costo debido a que este es un proyecto de graduación.

Tabla 3.13: Costos en colones asociados al desarrollo de la solución

No.	Descripción	Costo
1	Depreciación	320000
2	Diseño y desarrollo de solución	432000
	Total	752000

Aparte del ahorro en el desarrollo de la solución el laboratorio *LIANA* también se beneficia de las siguientes maneras:

- La base de datos de conocimiento en robótica cognitiva se incrementa. En especial, se genera nuevo conocimiento en cuanto a la implementación de sistemas robóticos en *ROS* 2 y el control asociado.
- El modelo físico reconstruido se implementa en el ambiente simulado para la ejecución de acciones atómicas lo cual queda disponible para experimentación futura.
- Como consecuencia de la realización del proyecto se puede formar un ingeniero más para la comunidad costarricense.

Comparativa de precios de solución

Es importante destacar el retorno económico que la solución tendrá para el laboratorio de forma cuantitativa. Para esto, se debe declarar el costo del robot utilizado. En este proyecto la implementación se lleva a cabo con herramientas de uso libre pero se puede estimar el costo final del robot y compararlo con otras opciones disponibles en el mercado. Para esta estimación es importante declarar la lista de componentes que se usan para construir el robot usado para la validación. En la tabla 3.14 se muestra la lista de componentes para implementar el modelo físico sumado al costo de desarrollar el proyecto:

Tabla 3.14: Costos de manufactura de robot de solución

No.	Componente	Cantidad	Precio (colones)	Total (colones)
1	Servomotor SG90 [54]	4	3046.4	12185.6
2	Arduino UNO [55]	1	9216	9216
3	Set de tornillos M3 [56]	20	35.84	716.8
4	Impresión 3D	1	137000	137000
5	Costos de desarrollo	1	752000	752000
			Total	911118.4

El coste total del robot incluye el desarrollo de la lógica de control en *ROS* 2. A continuación se compara el precio de sólo el robot con otros en el mercado para determinar una relación de costo. El costo de desarrollo se asume que será similar a las otras opciones. En la tabla se muestra el resumen de la comparación:

Tabla 3.15: Comparativa de precios de brazos robóticos

No.	Robot	Precio (colones)
1	Arduinobot [57]	159118.8
2	LynxMotion 3DOF Arm [58]	225555.19
3	Cyroni 3DOF Arm [59]	109437.96
4	Core Electronics 3DOF Arm [60]	37781.42
5	tztrobot 3DOF Arm [61]	347932.54

Para averiguar la relación de costo, basta con dividir el precio del robot que se desea comparar entre el precio del robot de referencia *arduinobot* que es el utilizado en este proyecto. A continuación se resume la lista de observaciones de comparación:

- El costo promedio en colones de los brazos robóticos 2, 3, 4 y 5 es de 180176.78. El robot usado en este proyecto es 13 % más barato que otras soluciones en el mercado. Una diferencia de 21057.98 colones.
- El robot de este proyecto se puede construir con componentes que se encuentran en el país mientras que las otras opciones robots que tendrían que ser comprados internacionalmente.
- Se destaca el caso del robot 4, que es el más barato de todos. Su costo no incluye el controlador del robot, por lo que queda a tarea del usuario escogerlo. El resto de robots tienen un controlador definido e incluido como parte de su ensamble.
- Los robots más caros, como el 2 y 5 tienen mucha mayor robustez en su chasis porque no usan impresión 3D debido a que están construidos en metal lo cual justifica su precio incrementado en comparación a la referencia.

En resumen, el robot escogido para este proyecto tiene la siguiente lista de retornos económicos para el laboratorio:

- Se ahorra 21057.98 colones con respecto a otras soluciones comerciales.
- Se reduce el tiempo de espera de compra de los componentes al escoger un robot que se puede manufacturar con componentes que se encuentran en Costa Rica.
- Al comparar el precio de la reconstrucción de *OSCAR* que son 470757.05 colones con el coste del robot de este proyecto se observa que esta alternativa de robot es 2.95 veces más barata para el laboratorio. Además, se puede construir con mayor velocidad debido al uso de componentes que se encuentran en el país.

Conclusión de análisis económico

La principal conclusión es que el dinero es el recurso más importante para este proyecto porque limita la reconstrucción del modelo físico de *OSCAR* y por lo tanto el método de validación usado para la solución. La falta de este recurso ocasiona que se deba implementar un plan de remedio en el cual se ahorre al máximo. Por medio del uso de tecnologías de uso libre se ahorra el costo del software necesario para la implementación y se puede simular el modelo del robot reconstruido.

Por último, se demuestra que el beneficio para el laboratorio en términos de dinero, conocimiento, formación de habilidades de ingeniería y implementación de tecnología de control en robótica cognitiva justifica el desarrollo de este proyecto.

Capítulo 4

Diseño de la solución

En este capítulo se describe paso a paso todos los procesos necesarios para implementar un sistema de robótica cognitiva en *ROS* 2. En la referencia [62] se demuestra un proceso de diseño para un robot manipulador. Este proceso de diseño se ajusta las necesidades de este proyecto y se muestra en la Fig. 4.1. Para cada uno de los procesos se describirán las tareas y observaciones principales. Debido a que el objetivo principal de esta solución es generar conocimiento sobre la implementación de *ROS* 2 se hará énfasis en detallar los comandos por usar y la estructura de programación.

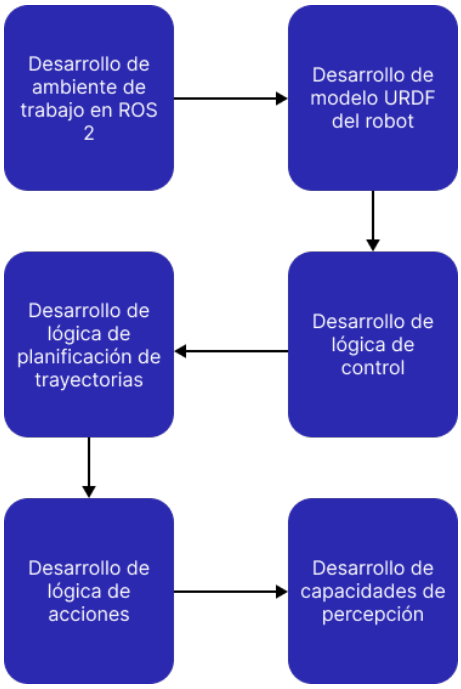


Figura 4.1: Diagrama de flujo para desarrollar la solución. Fuente: Elaboración Propia

4.1. Aspectos generales de la solución

Antes de comenzar con la programación del ambiente de robótica cognitiva se debe realizar un análisis de los fundamentos de la arquitectura robótica, modelo de robot y el conjunto de tecnologías que se ocupan. Para cada temática se busca destacar los elementos que forman parte de la solución.

4.1.1. Caracterización de arquitectura *OSCAR*

Si bien no se puede usar el modelo físico de *OSCAR* para validar el proyecto su arquitectura de *ROS* 1 es un recurso valioso. En la Fig. 4.2 se muestra la arquitectura cognitiva planteada inicialmente por el laboratorio. La arquitectura se encuentra obsoleta porque está implementada en *ROS* 1.

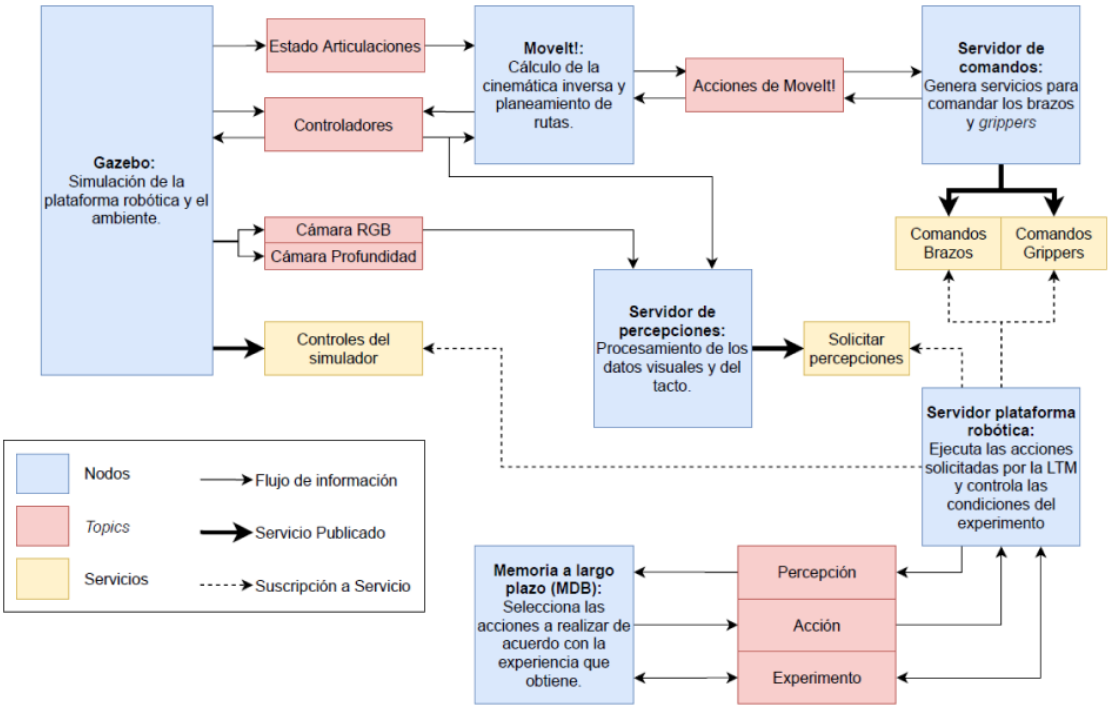


Figura 4.2: Arquitectura de ambiente de robótica cognitiva *OSCAR*. Fuente: [2]

A continuación se resume los principales aspectos técnicos de *ROS* 1 utilizados:

1. La arquitectura usa nodos para implementar el ambiente simulado, planificación de trayectorias y controlar actuadores.
2. La comunicación en tópicos se usa para conectar los nodos, transmitir información de sensores y estado de las articulaciones.

3. Se usan servicios para enviar los comandos de movimiento a los nodos que controlan el robot

Una observación importante es que la arquitectura aprovecha las unidades básicas de sistemas en *ROS* 1. Un cambio importante entre versiones es que mucha de la sintaxis queda deprecada. No obstante, los mismos componentes de *ROS* 1 también existen en *ROS* 2. Además se destaca que los cálculos de cinemática se resuelven con *MoveIt*, una biblioteca de *ROS* que facilita este aspecto importante en la operación de un robot. Otra observación es que se usa *Gazebo* para implementar un robot en un ambiente simulado. En general, la arquitectura obsoleta de *OSCAR* está diseñada para el control de sistemas robóticos de tipo manipulador. Por último, también se usa *ros control* para simular el control de los motores del robot.

4.1.2. Caracterización del robot a implementar

Debido a que no se cuenta con el modelo físico de *OSCAR* se debe escoger un modelo alternativo que permita validar las acciones atómicas a desarrollar. Como modelo alternativo se plantea usar otro robot de tipo manipulador nombrado Arduino bot [57]. Es un robot de 3 grados de libertad. Su controlador es un arduino. El robot utiliza servomotores para realizar los movimientos de *pitch*, *yaw* y *roll*. Este robot se escoge porque es un robot manipulador similar al modelo físico de *OSCAR* y el cliente lo considera adecuado para implementar las acciones atómicas. En la Fig. 4.3 se muestra una renderización del robot que se implementa.

Figura 4.3: Robot manipulador escogido para desarrollar. Fuente: [57]

Conocer los aspectos técnicos del robot es muy importante para describir su modelo en el formato descriptivo *URDF* en *ROS* 2. Para ese propósito se incluye el siguiente resumen técnico del robot:

1. El controlador del sistema es un arduino. Este tipo de controlador es útil para demostraciones de conceptos. Permite comunicación serial para comunicar el estado de los motores con programas de computación [63]. Es un controlador flexible que permite presentar una solución de nivel de prototipo pero funcional.

2. Se usan 4 servomotores SG90 [64]. 3 motores se usan para el movimiento de las articulaciones y 1 motor se usa para el movimiento del efector final del robot. Los motores son compatibles para ser controlados con el arduino.

3. El chasis del robot y los engranajes de transmisión de potencia son de uso libre y se pueden manufacturar usando impresión 3D en un material barato como el PLA.

4. El efector final del robot es una tenaza compuesta por dos dedos robóticos. El movimiento de los dedos es coordinado, es decir, se mueven al mismo tiempo. Se cierra y se abre por medio del uso de un servomotor SG90.

5. Cuenta con 3 eslabones y 4 articulaciones. 3 articulaciones conectan los eslabones. Sirven para los movimientos *pitch*, *yaw* y *roll*. La articulación restante corresponde a la conexión entre los eslabones y el efector final.

4.1.3. Conjunto de tecnologías

Para comenzar, en la tabla 4.1 se incluye una lista de las herramientas que se ocupan para esta implementación de control de un brazo manipulador en *ROS* 2. Para cada una de las tecnologías se provee una referencia que muestra la documentación de los programas para su instalación con el fin de lograr que este proyecto sea reproducible. Es importante destacar que todos los programas son de libre acceso y no tienen un costo por su instalación. Se recomienda revisar el código de la solución de este proyecto porque ahí se encuentran las bibliotecas necesarias para correr la solución con comentarios descriptivos.

Tabla 4.1: Lista de software necesario para implementar la solución

No.	Tecnología	Link Referencia
1	Ubuntu 22.04.3 LTS	[65]
2	ROS 2 Humble LTS	[66]
3	Gazebo Simulator	[67]
4	Visual Studio Code	[68]

Ahora que ya se tiene acceso a las herramientas se procederá a desarrollar la solución según las etapas de la Fig. 4.1.

4.2. Desarrollo de ambiente de trabajo en ROS 2

Para comenzar con cualquier proyecto de *ROS* 2 se debe establecer un espacio de trabajo. Para este proyecto el espacio de trabajo se nombra como *arduinobot_ws*. Se recomienda nombrar los espacios de trabajo en base al robot que se desea controlar. Preparar un ambiente de trabajo no es más que crear un folder en la computadora. Dentro de este folder se colocarán los paquetes que componen las acciones atómicas de la solución. El nombre se escoge siguiendo recomendaciones para nombres de proyectos en *ROS* 2 [69]. El beneficio que trae organizar de forma adecuada los proyectos en acciones atómicas *ROS* 2 es la modularidad. En este proyecto se establecen fundamentos de robots manipuladores que luego se pueden ajustar para que funcionen en el modelo físico de *OSCAR* cuando se haya reconstruido. El desarrollo de esta etapa es corto pero se documenta con el objetivo de aclarar el almacenamiento de proyectos en *ROS* 2.

4.3. Desarrollo de modelo URDF del robot

Inicialmente, se debe crear un paquete en el cual se contiene el modelo del robot. Se nombra este paquete *arduibot_description*. Ahora, es necesario describir en formato URDF el robot. Es bueno recordar que el *Unified Robot Description Format* es un archivo con extension *urdf.xacro*. Este formato aprovecha la estructura de programación XML en el cual los elementos del robot se describen por medio de etiquetas [70]. En el siguiente listing se muestran los elementos más importantes de estructura robótica en formato URDF, un eslabón y una articulación:

```
1
2  <!-- THIS LINK WILL ROTATE IN REFERENCE TO THE PARENT LINK THE BASE-->
3      <!-- THIS LINK ENABLES A YAW ROTATION -->
4      <link name = "base_plate">
5          <!-- THE INERTIA MACRO IS INITIALIZED WITH A MASS OF 0.1 OF BASE
       PLATE-->
6          <xacro:default_inertial mass="0.1"/>
7          <!-- VISUAL TAG IS SIMILAR BUT THE COORDINATES NEED TO CONSIDER
      PLACEMENT-->
8          <!-- FOR REFERENCE THE MODEL PLANS NEED TO BE REVIEWED-->
9          <visual>
10             <!-- COORDINATES ARE EXTRACTED FROM DESIGN DRAWINGS -->
11             <origin rpy = "0 0 0" xyz = "-0.39 -0.39 -0.56"/>
12             <geometry>
13                 <!-- THIS MESH IS THE PLATE THAT ROTATES-->
14                 <mesh filename = "package://arduinobot_description/
      meshes/base_plate.STL" scale = "0.01 0.01 0.01"/>
15             </geometry>
16         </visual>
17         <!-- THE COLLISION TAG DECLARES THE VOLUME OF THE LINK IN THE
      WORLD FOR COLLISION DETECTING-->
18         <!-- COLLISION PROPERTIES ARE DECLARED USING THE MESH-->
19         <collision>
20             <origin rpy = "0 0 0" xyz = "-0.39 -0.39 -0.56"/>
21             <geometry>
22                 <mesh filename = "package://arduinobot_description/
      meshes/base_plate.STL" scale = "0.01 0.01 0.01"/>
23             </geometry>
24         </collision>
25     </link>
26
27 <!-- NOW A REVOLUTE JOINT IS DESCRIBED-->
28     <joint name="joint_1" type="revolute">
29         <!-- THE PARENT IS THE FIXED BASE LINK-->
30         <parent link = "base_link"/>
31         <!-- THE CHILD IS THE FIRST LINK-->
32         <!-- ROBOT CONSTRUCTION STARTS FROM THE BASE AND MOVES UNTIL END
       EFFECTOR-->
33         <child link = "base_plate"/>
34         <!-- THE AXIS TAG INDICATES THE AXIS FOR REVOLUTE-->
35         <!-- IT IS A YAW MOVEMENT BECAUSE IT ROTATES IN Z AXIS-->
```

```
36          <axis xyz = "0 0 1"/>
37          <!-- THE ORIGIN TAG SETS THE POSITION OF THE LINKS BASED ON
    DRAWINGS-->
38          <origin rpy = "0 0 0" xyz="0 0 0.307"/>
39          <!--THESE ARE THE MECHANICAL LIMITS OF THE JOINT-->
40          <!--THEY ALLOW TO CONTROL THE MAXIMUM ROTATION ANGLE-->
41          <limit lower="-${PI/2}" upper="${PI/2}" effort="${effort}"
    velocity="${velocity}"/>
42       </joint>
```

Listing 4.1: Eslabón y articulación de ejemplo

Se recomienda la lectura de los comentarios porque explican el funcionamiento de cada etiqueta. En caso del eslabón que se define se tienen las siguientes características:

1. Tiene una masa de 0.1 kg. Se asignan propiedades de inercia constantes en todas las direcciones debido a la simpleza y simetría que tiene el eslabón.
2. Tiene propiedades visuales para mostrar una pieza STL en una escala de metros. En sus propiedades visuales se asigna la posición de la pieza en el marco de referencia.
3. Tiene propiedades de colisión que asignan el volumen del eslabón en el espacio.

En cuanto a la articulación se cuentan con las siguientes características:

1. Es una articulación de tipo rotacional.
2. Se debe establecer el eslabón padre, en este caso es la placa base. El eslabón hijo será la placa rotatoria del movimiento *yaw*, es decir, rotar en el eje z.
3. Los límites de rotación son 90.

Las características previas y la lógica asociada a cada una de ellas se aplica de forma iterativa en el resto de articulaciones y eslabones del robot. Las principales variables entre los eslabones y articulaciones son:

1. Posición del eslabón: Se deben revisar los planos del robot para establecer la posición del eslabón en el espacio.
2. Masa del eslabón: Se debe tomar en cuenta la masa de cada pieza que se ensambla.
3. Posición de la articulación: De igual forma, se debe colocar la articulación en la posición correcta para lograr una junta apropiada entre eslabones.
4. Tipo de articulación: Se debe tener claro si se requieren articulaciones fijas que no permitan movimiento o de rotación.

Con el fin de documentar debidamente los valores de posición y masa que se usan para cada uno de los eslabones de la programación se propone la tabla 4.2. Para establecer de forma clara la estructura de conexión en las articulaciones se desarrolla la tabla 4.3.

Tabla 4.2: Lista de eslabones del robot

No	Nombre	Posición x (m)	Posición y (m)	Posición z (m)	Masa (kg)
1	world	0	0	0	0
2	base link	-0.5	-0.5	0	1.0
3	base plate	-0.39	-0.39	-0.56	0.1
4	forward drive arm	0.19	0.06	-0.08	0.1
5	horizontal arm	-0.03	-0.4	-0.06	0.1
6	claw support	0	-0.05	-0.15	0.05
7	gripper right	-0.1	0.5	-0.1	0.01
8	gripper left	-0.04	-0.5	-0.1	0.01

Tabla 4.3: Lista de articulaciones en modelo URDF

No	Tipo	Nombre	Padre	Hijo	Áng. máx (°)	Áng. Mín(°)
1	Fija	virtual joint	world	base link	0	0
2	Rotacional	joint 1	base link	base plate	90	-90
3	Rotacional	joint 2	base plate	forward drive arm	90	-90
4	Rotacional	joint 3	forward drive arm	horizontal arm	90	-90
5	Fija	arm to claw support	horizontal arm	claw support	0	0
6	Rotacional	joint 4	claw support	gripper right	0	-90
7	Rotacional	joint 5	claw support	gripper left	0	-90

4.3.1. Visualización de modelo *URDF*

Para visualizar el modelo *ROS* de forma sencilla con un comando reutilizable se debe preparar un archivo de inicialización dentro del paquete del modelo *URDF*. El archivo de inicialización se encarga de habilitar la comunicación entre la herramienta de visualización

Rviz y publicar el estado de las articulaciones del robot. Esta herramienta es parte de *ROS* 2 y permite simular el funcionamiento del robot. A continuación se muestra el comando resultante que se puede usar para iniciar el módulo de visualización del robot:

```
ros2 launch arduinobot_description display.launch.py
```

Listing 4.2: Comando de inicialización de visualización del robot

Al utilizar el comando, se abre la ventana que se muestra en la Fig. 4.4 . Se observa que la herramienta de visualización permite interactuar con las articulaciones del robot que se han programado en su *URDF*. Por medio del uso de esta interfaz se observa que el rango de movimiento sea adecuado. Además, se verifica que el movimiento de la tenaza del efector final sea opuesto y sincronizado. En general, el movimiento del robot es adecuado y se considera listo para la siguiente etapa de desarrollo.

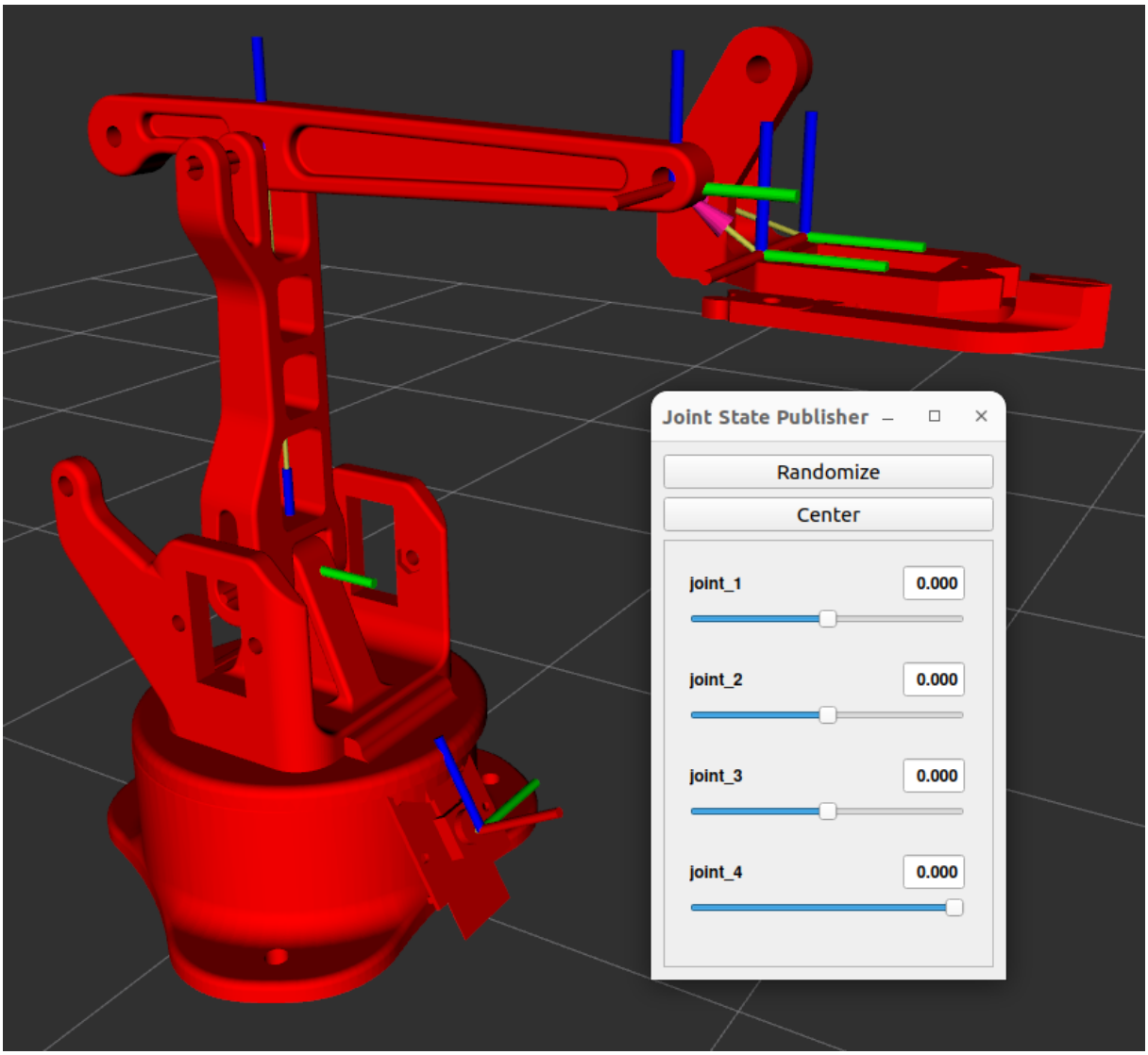


Figura 4.4: Fuente: Elaboración propia

De igual forma, se prepara un archivo de inicialización de la simulación en gazebo que

se llama *gazebo.launch.py*. Este archivo se encarga de inicializar el ambiente simulado del robot siguiendo las instrucciones declaradas en el *URDF*. El comando para inicializar la simulación es:

```
ros2 launch arduinobot_description gazebo.launch.py
```

Listing 4.3: Comando de inicialización de simulación del robot

4.4. Desarrollo de lógica de control

Ahora, se procede a explicar el desarrollo del paquete encargado de controlar los motores de cada articulación para lograr el movimiento adecuado del robot. El primer paso es crear un paquete de *ROS* 2 y nombrarlo como *arduinobot_ controller*. Luego de construir el paquete se deben ejecutar la siguiente lista de acciones para implementar un control de trayectorias de articulaciones [71]:

1. Definir propiedades de transmisión entre los motores y las articulaciones del robot en su modelo *URDF*.
2. Crear archivo de descripción de control de articulaciones
3. Implementación de archivo de simulación de control
4. Desarrollar archivo de descripción de controlador
5. Inicialización de controlador en arquitectura

4.4.1. Propiedades de transmisión entre los motores y las articulaciones del robot

Esta operación se realiza en el archivo de descripción *URDF* ya desarrollado. En el siguiente listing se muestran las propiedades de transmisión de potencia que hay entre los motores y las articulaciones:

```

</xacro:macro>
    <!-- TRANSMISSION PROPERTIES FOR THE MOTORS AND THE JOINTS IN THE ROBOT-->
    <xacro:macro name = "default_transmission" params = "number">
        <transmission name = "transmission_${number}">
            <plugin>transmission_interface/SimpleTransmission</plugin>
            <!-- THIS LINE DECLARES THE SG90 SERVO ACTUATORS-->
            <actuator name = "motor_${number}" role = "actuator1"/>
            <joint name = "joint_${number}" role ="joint1">
                <!-- THE MECHANICAL REDUCATION REPRESENTS THE LOSS IN THE TRANSMISSION-->
```

```
                <mechanical_reduction>
                    1.0
                </mechanical_reduction>
            </joint>

        </transmission>
    </xacro:macro>
```

Listing 4.4: Eslabón y articulación de ejemplo

Las principales observaciones acerca de la implementación son:

1. Se define un *macro* para utilizar las mismas propiedades en todas las articulaciones porque se usa el mismo tipo de motor en todas. De esta forma se reduce la cantidad de código por desarrollar.

2. La reducción mecánica se representa como unitaria para indicar al sistema simulado que no hay perdida de potencia entre los motores y las articulaciones.

3. Se usa el módulo de transmisión simple [72] de *ROS* 2.

4.4.2. Descripción de control de articulaciones

En el paquete de descripción *URDF*, se coloca el archivo *arduinobot_ros2_control.xacro*. En este archivo se describen los límites físicos de las articulaciones al controlador. Se siguen los mismos límites ya establecidos en la tabla 4.3. A continuación, se resalta un listing con el formato general de una articulación con el formato adecuado para ser reconocido por el controlador.

```
<!-- HERE THE JOINT FOR THE RIGHT GRIPPER IS DEFINED-->
        <joint name = "joint_5">
            <!-- THIS IS HOW TO MIMIC THE JOINT MOVEMENT-->
            <param name = "mimic">joint_4</param>
            <!-- THIS IS HOW TO INVERT THE MOVEMENT-->
            <param name = "multiplier">-1</param>
            <!-- THE CONTROLLER IS FOR THE POSITION-->
            <command_interface name = "position">
                <!-- DECLARE THE PHYSICAL LIMITS OF THE JOINT-->
                <param name ="min">0.0</param>
                <param name = "max">${PI/2}</param>
            </command_interface>
        </joint>
```

Listing 4.5: Declaración de articulación en lógica de controlador

En este ejemplo, se programa el dedo derecho de la tenaza del efector final. La articulación se mueve forma sincronizada con el dedo izquierdo pero en dirección opuesta [73].

4.4.3. Archivo de simulación Gazebo

De igual forma, en el paquete de descripción del robot se crea un archivo nombrado *arduinobot_gazebo.xacro*. En este archivo se declara el controlador en el ambiente de simulación *Gazebo*. Es una declaración corta, donde el principal aspecto a destacar es que se aprovecha el uso de los controladores disponibles como parte del kit de desarrollo de software *ROS* 2. En el siguiente listing se muestra como declarar un controlador de forma simulada:

```
<!-- DECLARATION FOR THE CONTROLLER IN GAZEBO-->
   <gazebo>
       <!-- USE THE ROS2 CONTROL LIBRARY-->
       <plugin filename ="libgazebo_ros2_control.so" name ="
  gazebo_ros2_control">
           <!-- ASSIGN THE ROBOT DESCRIPTION AS A PARAMETER-->
           <robot_param>robot_description</robot_param>
           <!-- ASSIGN TOPIC COMMUNICATION-->
           <robot_param_node>robot_state_publisher</robot_param_node>
           <parameters>$(find arduinobot_controller)/config/
  arduinobot_controllers.yaml</parameters>
       </plugin>
   </gazebo>
```

Listing 4.6: Declaración de control en ambiente de simulación

4.4.4. Archivo de descripción de controlador

En el paquete de control se encuentra el archivo nombrado *arduinobot_controllers.yaml*. Es un archivo de extensión *YAML* y sirve para declarar el tipo de controlador que se va a usar para el robot [74]. También se declaran dos grupos de articulaciones que se van a controlar, las del chasis del robot y las de la tenaza. Ambos controladores son de la biblioteca de control de articulaciones de *ROS* 2. Esta biblioteca es adecuada para controlar el brazo robótico por implementar debido a su simpleza. A continuación se muestra un listing en el cual se definen los controladores y los grupos de articulaciones por controlar:

```
<!-- DECLARATION FOR THE CONTROLLER IN GAZEBO-->
   <gazebo>
       <!-- USE THE ROS2 CONTROL LIBRARY-->
       <plugin filename ="libgazebo_ros2_control.so" name ="
  gazebo_ros2_control">
           <!-- ASSIGN THE ROBOT DESCRIPTION AS A PARAMETER-->
           <robot_param>robot_description</robot_param>
           <!-- ASSIGN TOPIC COMMUNICATION-->
           <robot_param_node>robot_state_publisher</robot_param_node>
           <parameters>$(find arduinobot_controller)/config/
  arduinobot_controllers.yaml</parameters>
       </plugin>
```

```
    </gazebo>
```

Listing 4.7: Declaración de controladores robóticos

4.4.5. Inicialización de controlador

Para lograr una inicialización modular del controlador se requiere desarrollar un archivo de inicialización. Este archivo de inicialización se coloca en el paquete de control y se llama *controller.launch.py*. Se encarga de habilitar el módulo de control listando los documentos previamente desarrollados. Se puede decir que el archivo de inicialización concluye el desarrollo del controlador del robot. En el siguiente listing se destacan los elementos principales de este código:

```
return LaunchDescription([
        robot_state_publisher,
        joint_state_broadcaster_spawner,
        arm_controller_spawner,
        gripper_controller_spawner
    ])
```

Listing 4.8: Inicialización del paquete de controlador

4.5. Desarrollo de lógica de planificación de trayectorias

En esta sección se describe el paquete que se encarga de resolver la cinemática del robot al realizar movimientos en su espacio de trabajo. El nombre del paquete es *arduinobot_moveit*. En este paquete se aprovecha la biblioteca *MoveIt* que se encarga de resolver los cálculos necesarios para mover la tenaza a cualquier posición. Además se implementan excepciones en caso de que la posición de destino no se encuentre dentro de los límites físicos del robot. La lista de tareas para implementar este paquete son:

1. Preparar servicio para conversión de cuaternión a ángulos de euler
2. Configurar el planificador de trayectorias *MoveIt*
3. Desarrollar el archivo de inicialización del paquete

4.5.1. Servicio de conversiones angulares de euler y cuaterniones

El objetivo de este servicio es facilitar la conversión de las representaciones angulares de euler que son entendibles por los humanos a cuaterniones que son fáciles de procesar en las operaciones matriciales. Se plantea como servicio debido a que es una tarea repetitiva pero

que tiene entrada variable según la posición que se desea alcanzar. Un servidor funciona en base a solicitudes y respuestas. En el siguiente listing se muestra la solicitud y respuesta de conversión de cuaternión a grados euler:

```
 #REQUEST FOR THE SERVER
#INPUTS THE QUATERNION COMPONENTS
float64 x
float64 y
float64 z
float64 w
---

#RESPONSE FOR THE SERVER
#OUTPUTS THE EULER ANGLES
float64 roll
float64 pitch
float64 yaw
```

Listing 4.9: Solicitud y respuesta de conversión angular

Las solicitudes y respuestas de los servicios se colocan en un paquete nombrado *arduinobot_msgs*. Es en este paquete donde se define el formato de la mensajería del servicio. En este caso, los componentes de los angulos son numeros decimales. Los servicios disponibles se encuentran en los archivos *EulertoQuaternion.srv* y *QuaterniontoEuler.srv* respectivamente.

Ahora que la mensajería ya está descrita, se procede a implementar el servidor. Se crea otro paquete, nombrado *arduinobot_utils* donde se coloca un programa que se llama *angle_conversion.py*. Dentro de este programa se destaca la lógica en la cual se recibe de entrada cuaterniones o ángulos de euler y por medio del uso de la biblioteca de transformaciones de ángulos de *ros* 2 se resuelve la operación matemática [75]. El siguiente listing muestra la lógica de transformación:

```
# THIS FUNCTION RECEIVES A REQUES IN EULER AND RETURNS QUATERNIONS
    def eulerToQuaternionCallback(self, req, res):
        # THE POSITION IS DECLARED BASED ON PITCH YAW AND ROLL ANGLES
        self.get_logger().info("Requested to convert euler angles roll: %f, pitch: %f, yaw: %f, into a quaternion." % (req.roll, req.pitch, req.yaw))
        # BY USING THE TF LIBRARY THE INPUT IS TRANSFORMED
        (res.x, res.y, res.z, res.w) = quaternion_from_euler(req.roll, req.pitch, req.yaw)
        # MESSAGES ARE PRINTED WITH THE RESPONSE
        self.get_logger().info("Corresponding quaternion x: %f, y: %f, z: %f, w: %f" % (res.x, res.y, res.z, res.w))
        return res
    # SIMILAR LOGIC BUT INVERSE TO ALLOW THE OTHER TRANSORMATION
    def quaternionToEulerCallback(self, req, res):
        # COMMUNICATION IS SENT WHEN A REQUEST IS RECEIVED
        self.get_logger().info("Requested to convert quaternion: %f, y: %f, z: %f, w: %f" % (req.x, req.y, req.z, req.w))
```

```
14          (res.roll, res.pitch, res.yaw) = euler_from_quaternion([req.x,
    req.y, req.z, req.w])
15          self.get_logger().info("Corresponding euler angles roll: %f,
    pitch: %f, yaw: %f" % (res.roll, res.pitch, res.yaw))
16          return res
```

Listing 4.10: Lógica de transformación angular

4.5.2. Configuración de planificador de trayectorias *MoveIt*

Ya se puede comunicar facilmente por medio de un servicio las posiciones que se deben alcanzar en cuaterniones y convertirlas de vuelta a ángulos de euler. Ya se puede configurar el paquete de cinemática de trayectorias *MoveIt* [76]. Dentro del paquete correspondiente, se debe crear la siguiente lista de archivos:

1. *arduinobot.srdf*: En este archivo se declaran los grupos de control. En este caso se plantean dos grupos, el del brazo y el de la tenaza. Además se establecen propiedades de colisión en las piezas del robot.

2. *initial_positions*: Aquí se definen la posición inicial de las articulaciones. Para este robot, la posición de inicio es 0 *pitch*, 0 *yaw* y 0 *roll*. Además la tenaza se deja cerrada.

3. *joint_limits*: Sirve para establecer los límites físicos de movimiento de las articulaciones.

4. *kinematics*: Se escoge la biblioteca de resolución de cinemática. En este caso se aprovecha la biblioteca incluida en *ROS* 2 que se llama *KDLKinematicsPlugin*.

5. *moveit_controllers*: En este archivo se declara el controlador que se encargara de enviar las señales a los motores para ejecutar la ruta solicitada. Se conecta con el controlador desarrollado previamente.

6. *moveit.rviz*: En este archivo se guarda la configuración de *Rviz* para que funcione con la integración de *MoveIt* para que cada vez que se abra la herramienta de visualización el módulo de trayectorias se cargue automáticamente.

7. *pilz_cartesian_limits*: En este archivo se declaran los límites de velocidad del robot. Para este proyecto no se solicita control de velocidad por lo que se asigna un valor unitario a la velocidad de 1 metro por segundo.

   En general, configurar el planificador de trayectorias involucrar la creación de 7 archivos en su paquete [76]. Es un proceso para el cual se requiere de cuidado y paciencia.

4.5.3. Inicialización del paquete de trayectorias en arquitectura

De igual forma los paquetes anteriores este también necesita un archivo de inicialización para integrarlo a la arquitectura. El archivo inicializador se llama *moveit.launch.py*. Tiene una estructura similar al resto de inicializadores. El comando que se encarga de lanzar el módulo es:

```
ros2 launch arduinobot_moveit moveit.launch.py
```

Listing 4.11: Comando de inicialización de módulo de trayectorias

Al ejecutar el comando se abre la herramienta de visualización de trayectorias que se muestra en la Fig. 4.5. Al dar clic y arrastrar en la tenaza se puede mover el robot usando cinemática inversa y en caso de dar clic y arrastrar las articulaciones se hace uso de la cinemática directa. Su superficie se torna roja en caso de alcanzar alguna posición que implique colisión. El usuario puede planificar trayectorias por medio de su uso.

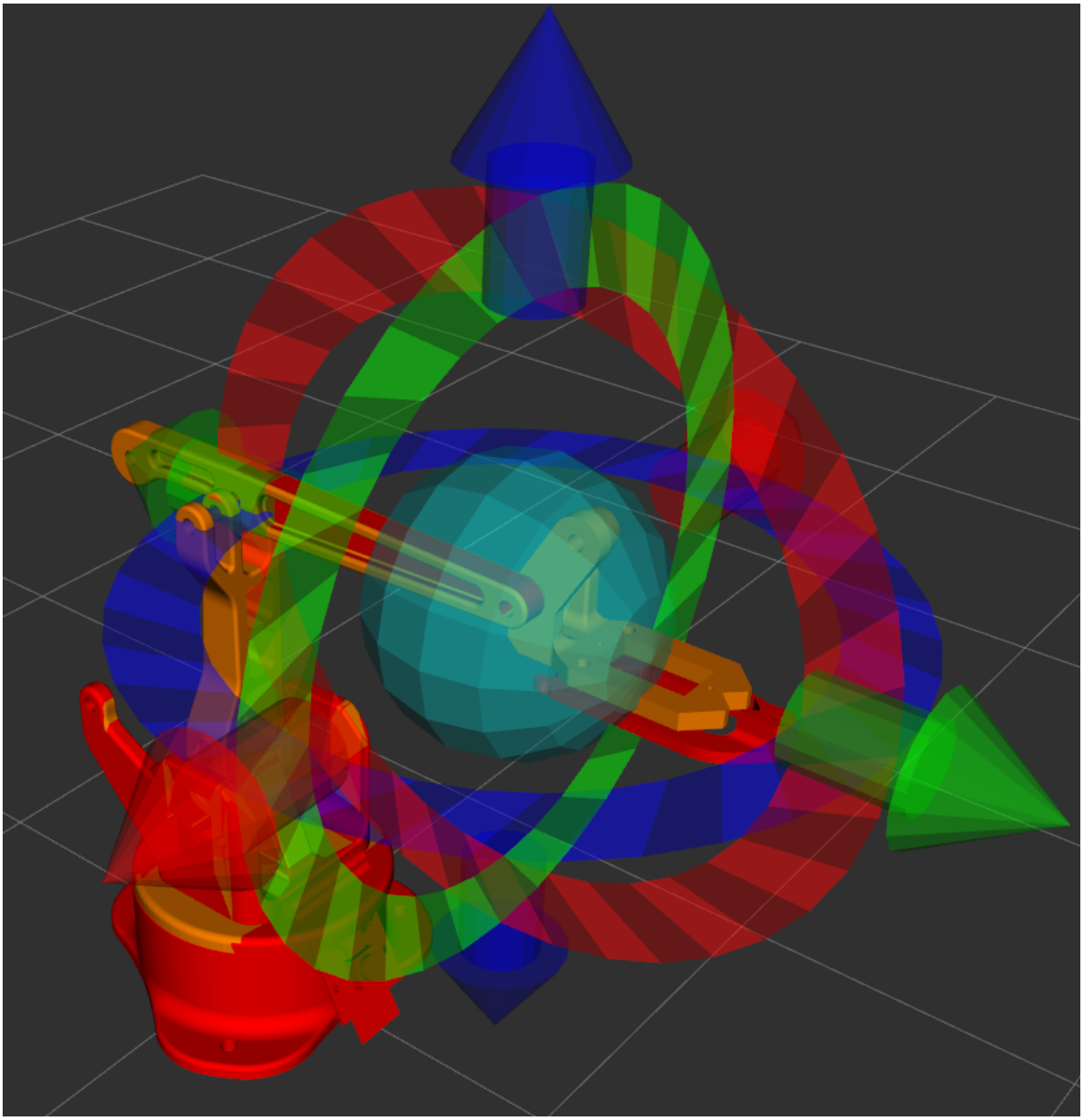


Figura 4.5: Fuente: Elaboración propia

Ahora se ha completado el proceso de descripción del módulo encargado de calcular y planificar las trayectorias del sistema. Se puede proceder a configurar el siguiente módulo.

4.6. Desarrollo de lógica de acciones atómicas de movimiento

Hasta este punto del desarrollo, el robot es simulable y ya es capaz de hacer los cálculos de movimiento. Ahora se puede plantear un servidor de acciones que se encargará de recibir un objetivo y retornar una respuesta con las instrucciones de movimiento para el

robot que se enviarán al módulo de trayectorias para luego ser simulado. **Estas serán las acciones atómicas que el robot puede realizar.** Esta etapa del desarrollo depende de todos los módulos desarrollados previamente.

4.6.1. Servidor de acciones

El servidor de acciones de movimiento se encarga enviar las instrucciones de movimiento a los controladores del robot para alcanzar una pose específica. Para este servidor hay 3 poses objetivo disponibles:

1. Pose de descanso: El robot se coloca en esta pose cuando se apaga.
2. Pose de espera: El robot se coloca en esta pose antes de iniciar su trabajo.
3. Pose objetivo: El usuario indica al robot una posición a alcanzar con la tenaza.

Es importante describir en el paquete de mensajería *arduinobot_msgs* la solicitud y respuesta del servidor de acciones. En el siguiente listing se observa la estructura de la mensajería del servidor de acciones:

```
#GOAL MESSAGE
int32 task_number
---
#RESULT MESSAGE
bool success
---
#FEEDBACK MESSAGE
int32 percentage
```

Listing 4.12: Estructura de mensajes de solicitud, respuesta y realimentación de acciones

La principal observación es que el objetivo es un número de tarea y el resultado es un indicador booleano de éxito o fallo en caso de que alguna trayectoria no sea válida. Una vez se define el mensaje, se procede a crear el paquete *arduinobot_remote*. Se nombra de esta forma porque en este paquete se colocan la interfaz de interacción humano máquina. El archivo *task_server.cpp* contiene la lógica de movimiento del robot para alcanzar las poses establecidas.

El objetivo del siguiente listing es mostrar la forma en la cual se indica al robot la pose a alcanzar:

```
// THESE COMMANDS USE MOVEIT API TO SEND A INSTRUCTION TO THE GROUPS
   THAT EXIST IN THE ROBOT
    auto arm_move_group = moveit::planning_interface::MoveGroupInterface
   (shared_from_this(), "arm");
    auto gripper_move_group = moveit::planning_interface::
   MoveGroupInterface(shared_from_this(), "gripper");

    std::vector<double> arm_joint_goal;
```

```
6     std::vector<double> gripper_joint_goal;
7
8     // THESE CONDITIONS CONTAIN THE OBJECTIVE
9     // THIS ONE IS TO GO TO ACTIVATE POSITION
10    if (goal_handle->get_goal()->task_number == 0)
11    {
12      arm_joint_goal = {0.0, 0.0, 0.0};
13      gripper_joint_goal = {-0.7, 0.7};
14    }
15    // THIS CONDITION IS TO GO TO GRIP GOAL POSE
16    else if (goal_handle->get_goal()->task_number == 1)
17    {
18      arm_joint_goal = {-1.14, -0.6, -0.07};
19      gripper_joint_goal = {0.0, 0.0};
20    }
21    // THIS ONE IS TO GOT TO SLEEP POSE
22    else if (goal_handle->get_goal()->task_number == 2)
23    {
24      arm_joint_goal = {-1.57,0.0,-0.9};
25      gripper_joint_goal = {0.0, 0.0};
26    }
```

Listing 4.13: Comunicación de objetivos de movimiento de articulaciones del servidor de acciones

Se aprovecha el *API* de *MoveIt* para determinar si la trayectoria es válida. La pose objetivo se indica en términos de *roll*, *pitch* y *yaw* para las articulaciones del brazo y en en ángulo de apertura para la tenaza del efector final. Las 3 poses objetivo se encuentran descritos en términos de ángulos de euler.

De esta forma se concluye el desarrollo del servidor de acciones. Ahora se puede enviar una solicitud a este servidor para obtener en su respuesta los ángulos de euler para alcanzar cualquiera de las 3 poses implementadas.

4.7. Desarrollo de capacidades de percepción

Para este robot, se plantea el control por medio de comandos de voz comunicados a través de *Alexa* la asistente de voz de *Amazon* [46]. Por medio del uso del *API* de la plataforma de *Alexa* se puede asociar comandos de voz al servidor de acciones para indicar que se debe alcanzar cualquiera de las 3 poses establecidas. El flujo de trabajo es el siguiente:

1. Preparación de aplicación de *Alexa*
2. Desarrollo de modelo de interacción por voz
3. Inicialización de modelo de robot simulado

4.7.1. Preparación de aplicación de *Alexa*

Para comenzar, se crea una cuenta de desarrollador de *Alexa* de uso gratuito. Dentro de esta cuenta se desarrolla una habilidad de detectar 3 categorías de comandos que corresponden a alcanzar la pose objetivo espera y descanso. Esta operación se realiza en la consola de desarrollo de aplicaciones *Alexa* siguiendo su formulario de preparación. Se destaca la Fig. porque se muestra el campo de información donde se indica la dirección a la cual se comunicará el *API* de *Alexa* con el sistema robótico diseñado.

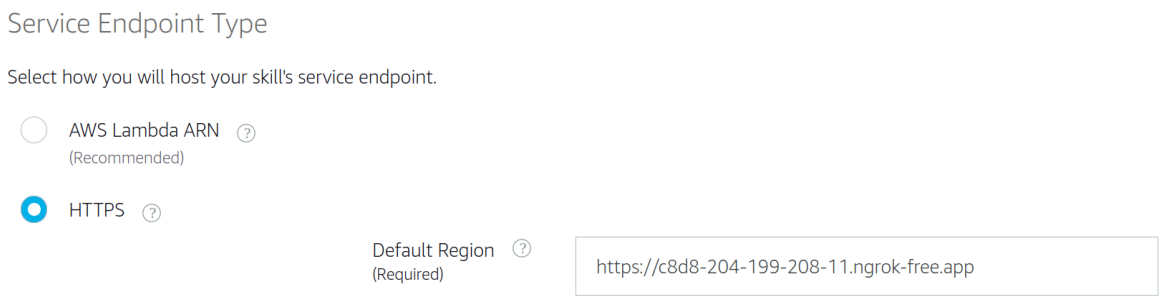


Figura 4.6: Fuente: Elaboración propia

Dicha dirección se genera por medio del uso de *ngrok* y se abre en el puerto 5000 de la red local de la computadora. Debido a que se usa una licencia libre de esta herramienta, la dirección cambia cada vez que se inicia la aplicación por lo que se debe tener el cuidado de reemplazarla en cada inicialización del robot. De esta forma ya se ha habilitado el *API* de *Alexa* para ser implementado con el sistema.

4.7.2. Desarrollo de modelo de interacción por voz

Ahora se hace un programa computacional encargado de traducir una instrucción recibida por medio del *API* de *Alexa* a un objetivo para el servidor de acciones y de esta forma activar el movimiento del robot. Dicha función se coloca en el paquete *arduinobot_remote* y se nombra *alexa_interface*. Por medio de clases se define cada uno de las poses objetivo para que se puedan invocar de forma recursiva. El siguiente listing muestra la estructura del comando de activación:

```
# THIS CLASS IS TO TAKE THE INPUT FROM VOICE COMMAND AND TURN INTO ACTION GOAL
class LaunchRequestHandler(AbstractRequestHandler):
    # THE HANDLE IS CATEGORIZED INTO THE LAUNCH REQUEST INTENT
    def can_handle(self, handler_input):
        return is_request_type("LaunchRequest")(handler_input)

    def handle(self, handler_input):
        # IF THE INTENT IS FOR ACTIVATION RETURN THE READY TO WORK MESSAGE
        speech_text = "Hi, I'm ready to work"
```

```
        handler_input.response_builder.speak(speech_text).set_card(
            SimpleCard("Online", speech_text)).set_should_end_session(
            False)
        # ASSIGN THE GOAL AS THE 0 NUMBER TO INDICATE TO TASK SERVER THE CORRECT GOAL
        goal = ArduinobotTask.Goal()
        goal.task_number = 0
        action_client.send_goal_async(goal)

        return handler_input.response_builder.response
```

Listing 4.14: Lógica de asocie de comando de voz con servidor de acciones

Siguiendo esta misma lógica se desarrollan los comandos de voz para las otras 2 poses. Además en esta programación se declara el número identificador de conexión *API* que conecta el código con la aplicación *Alexa* implementada previamente. En resumen, se hace un mapeo de las categorías de comandos de voz declaradas y las instrucciones de movimiento del servidor de acciones.

4.7.3. Inicialización de modelo simulado de robot

Este es el último paso de desarrollo de la solución. Consiste en preparar un nuevo paquete nombrado *arduinobot_ bringup* dentro del cual se crea el archivo *simulated_ robot.launch.py*. El propósito de este programa es inicializar todos los módulos de la arquitectura con un único comando que se muestra a continuación:

```
ros2 launch arduinobot_bringup simulated_robot.launch.py
```

Listing 4.15: Comando para iniciar sistema de robot simulado

Al ejecutar el comando se abren todos los módulos y el sistema se encuentra listo para ser utilizado como se muestra en la Fig. 4.7.

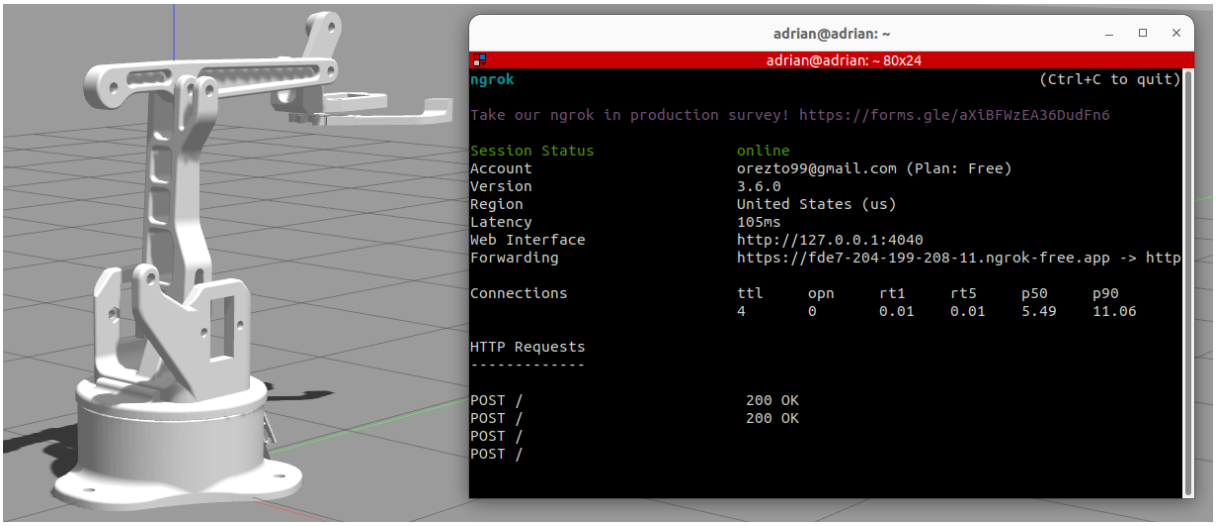


Figura 4.7: Fuente: Elaboración propia

De esta forma se concluye el desarrollo de la solución porque ya todos los módulos del sistema se han desarrollado y se han integrado en una arquitectura de robótica en *ROS* 2.

4.8. Caracterización final de la arquitectura implementada

Ahora que ya se han implementado todos los módulos del sistema es posible observar la arquitectura implementada de forma gráfica usando el comando del siguiente listing luego de ejecutar la inicialización del sistema [77]:

```
rqt_graph
```

Listing 4.16: Comando para graficar arquitectura del sistema

La arquitectura implementada se muestra en la Fig. 4.8. Se aprecia que el mismo sistema *ROS* 2 es capaz de detectar los nodos establecidos, el servicio de conversión angular y el servidor de acciones de movimiento. El diagrama muestra en detalle el flujo de información entre los nodos.

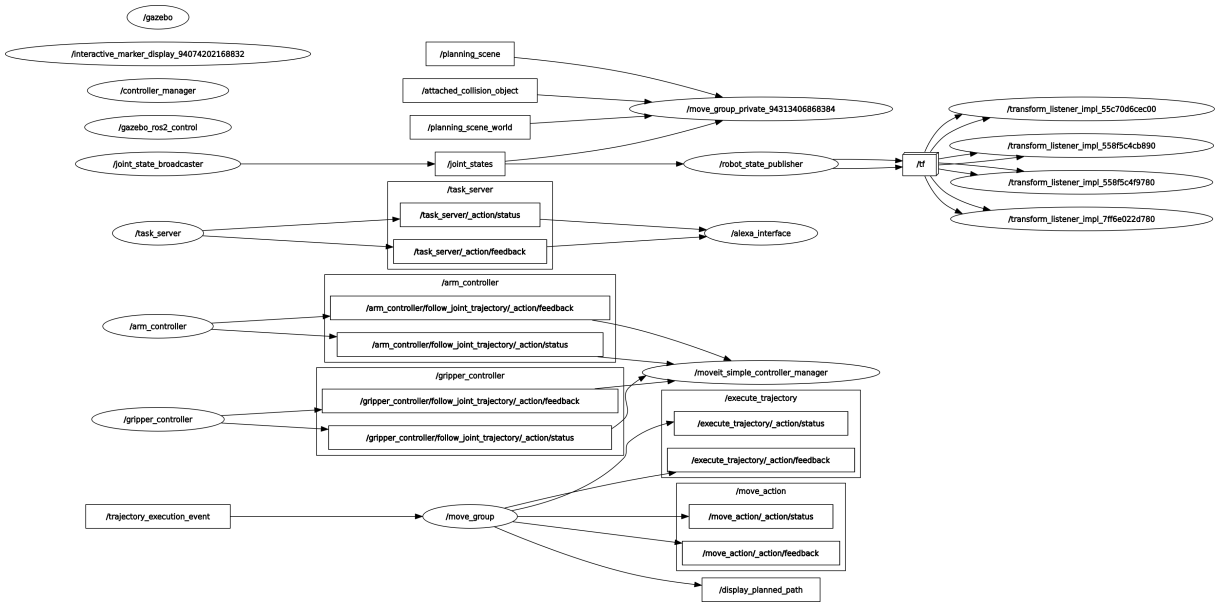


Figura 4.8: Fuente: Elaboración propia

Con el fin de presentar una versión más fácil de entender se presenta el diagrama de la Fig. 4.9. De esta forma es como un sistema de robótica cognitiva para manipuladores se implementa en *ROS* 2. En el siguiente capítulo se estudian los resultados del sistema.

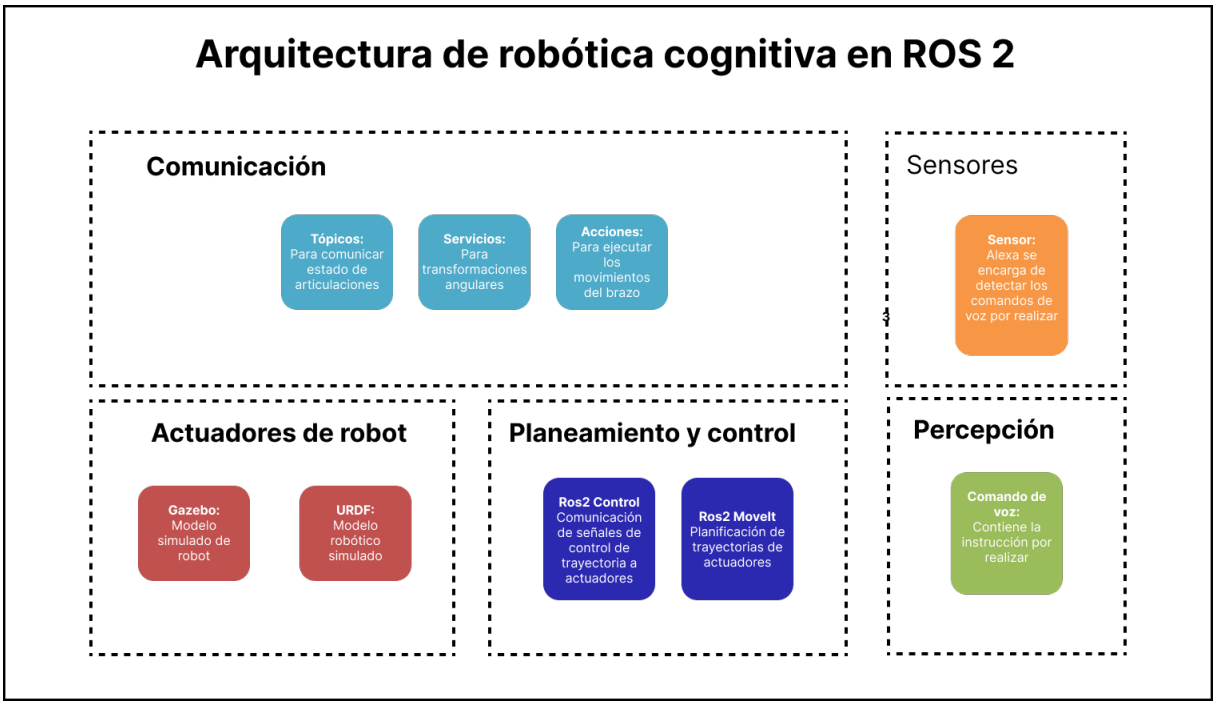


Figura 4.9: Fuente: Elaboración propia

Capítulo 5

Resultados y análisis

Ahora que el sistema es funcional, se debe estudiar los resultados que producen las pruebas de concepto planteadas. De esta forma se puede determinar cuantitativamente si las métricas objetivo se cumplen.

5.1. Primer experimento: Simulación de acciones atómicas de movimiento

En este experimento, se busca demostrar que las acciones atómicas implementadas funcionan dentro del ambiente de simulación. El flujo de trabajo para utilizar las acciones es el siguiente:

1. Inicializar ambiente de simulación
2. Enviar comando de voz al robot
3. Supervisar movimiento de simulación
4. Grabar proceso para documentar resultados

5.1.1. Resultados experimentales

Al ejecutar las pruebas, se envían los comandos de voz a través de la plataforma del asistente de voz. En la Fig. 5.1 se muestra que el asistente de voz recibe el comando y retorna una respuesta indicando el movimiento a realizar:

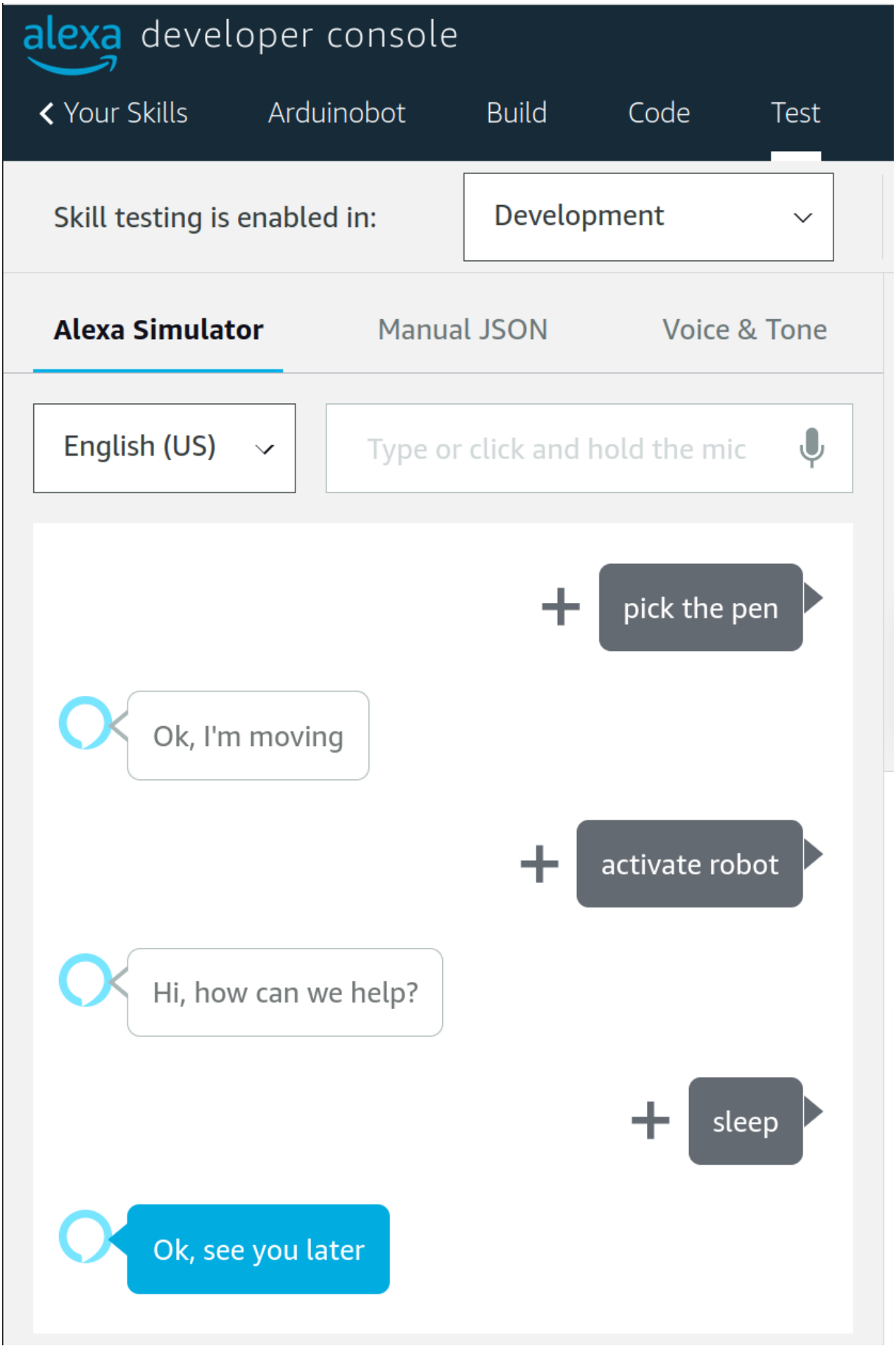


Figura 5.1: Fuente: Elaboración propia

Luego de enviar el comando de voz, se espera a que el sistema se encargue de interpretar la instrucción, obtener los comandos de movimiento del servidor de acciones, comunicar los comandos al controlador y por último observar el movimiento del robot para alcanzar el objetivo. En la tabla 5.1 se muestran videos evidenciando el correcto funcionamiento de cada comando:

Tabla 5.1: Resultados experimentales de acciones atómicas

Acción Atómica	Estado	Link de video
Pose de Descanso	Cumple	Link
Pose de Trabajo	Cumple	Link
Pose Objetivo	Cumple	Link

5.1.2. Análisis de resultados

Los resultados obtenidos para esta prueba confirman que el robot es capaz de alcanzar cualquiera de las 3 poses objetivo de las acciones atómicas diseñadas. Algunas observaciones son:

1. El robot no puede alcanzar la posición de descanso si se encuentra en la pose objetivo para restringir la interrupción de acciones durante su uso.

2. Los comandos de voz se logran interpretar de forma adecuada con el asistente alexa y el servidor de acciones porque se alcanzan las posiciones establecidas.

3. Se observa en base a los videos, que el tiempo por instrucción es cercano a 1 minuto. Si bien el tiempo de ejecución de cada instrucción no es instantáneo, es lo suficientemente rápido para ahorrar tiempo de programación al usar el robot. No se considera necesario tomar datos exactos de tiempo porque el cliente no está interesado en esta métrica, sólo está interesado en el correcto funcionamiento de cada acción.

De esta forma, se concluye que el experimento es exitoso y por lo tanto el concepto cumple con las necesidades del cliente.

5.2. Segundo experimento: Precisión de rotación

En este experimento, se estudia la precisión de movimiento que el sistema simulado tiene. Al analizar la diferencia entre la posición objetivo y la posición alcanzada se puede determinar si la solución cumple con el umbral de las especificaciones.

5.2.1. Resultados experimentales

Se desea averiguar con cuanta precisión se alcanza una posición objetivo en la simulación del sistema. Para esto, se plantea efectuar 10 pruebas con cada una de las poses que el robot tiene: descanso, trabajo y objetivo. Usando la interfaz del simulador se puede obtener el ángulo de rotación de cada una de las articulaciones. En la Fig. 5.2 se muestra la forma en la cual se recopilan los datos de posición consultando la información de posición por medio del simulador:

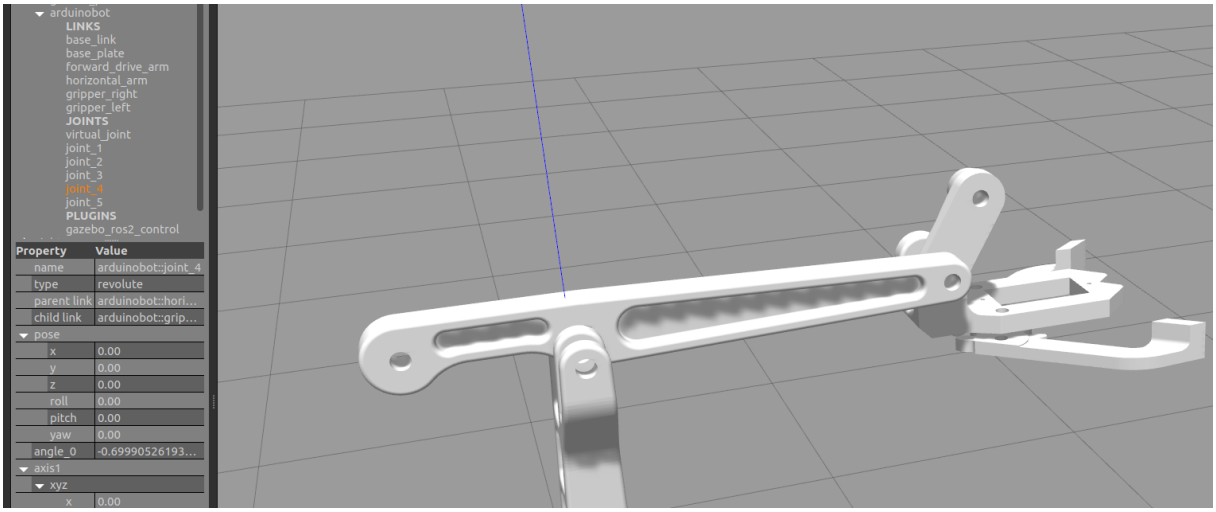


Figura 5.2: Fuente: Elaboración propia

Al efectuar 10 mediciones para cada pose del robot se llegan a los resultados experimentales de la tabla 5.2.

Tabla 5.2: Resultados experimentales de experimento de precisión de movimiento

Pose	Promedio (rad)	Objetivo (rad)	Diferencia (rad)	Desv. Est
Descanso				
roll	-1.563	-1.57	0.007	0.005
pitch	0	0	0	0
yaw	-0.892	-0.9	0.008	0.004
gripper	0	0	0	0
Espera				
roll	0.003	0	0.003	0.005
pitch	0	0	0	0
yaw	0.001	0	0.01	0.03
gripper	-0.693	-0.7	0.007	0.005
Objetivo				
roll	-1.134	-1.14	0.006	0.008
pitch	-0.595	-0.6	0.005	0.007
yaw	-0.7	-0,7	0	0
gripper	0	0	0	0

5.2.2. Análisis de resultados

A continuación las principales observaciones de los resultados del experimento:

1. La mayor diferencia de precisión se encuentra en el movimiento de yaw cuando el robot se dirige a su pose de descanso. En este caso, la diferencia es de 0.008 radianes, lo cual corresponde a 0.45 . El umbral de precisión de las especificaciones es de al menos 1 . En este caso se puede concluir **el concepto cumple con el valor objetivo**.

2. Los resultados experimentales presentan muy poca variación entre las mediciones como se refleja en las desviaciones estándar. Se observa que el resultado de mayor variación se da en la articulación de *roll* cuando el robot va a la posición objetivo.

3. Debido a que se usa un ambiente simulado, se observa que hay resultados perfectos, es decir, sin variación alguna como en el caso de la tenaza (*gripper*) en su posición de descanso y objetivo. En ambos casos, la tenaza se cierra por completo, lo cual el sistema logra sin diferencia. El sistema es muy exacto en estos casos.

4. El robot cumple con el valor ideal de umbral de precisión para cada una de las articulaciones del sistema por lo que se concluye que este experimento es exitoso.

5.3. Tercer experimento: Prevención de colisiones

Para este experimento, el sistema debe alcanzar 5 posiciones válidas y 5 posiciones inválidas. Se espera que el comportamiento de cada prueba demuestre que el robot puede determinar si sus límites físicos le permiten llegar a la pose establecida o si ocurrirá alguna colisión con su chasis.

5.3.1. Resultados experimentales

Para escoger las poses se hace uso de la herramienta de planificación de trayectorias del sistema. En la Fig. 5.3 se muestra la configuración con la que se pueden escoger trayectorias aleatorias válidas que el sistema debe alcanzar. Se observa que el cuerpo del robot se muestra en naranja cuando la pose por alcanzar si es válida y no implica colisión.

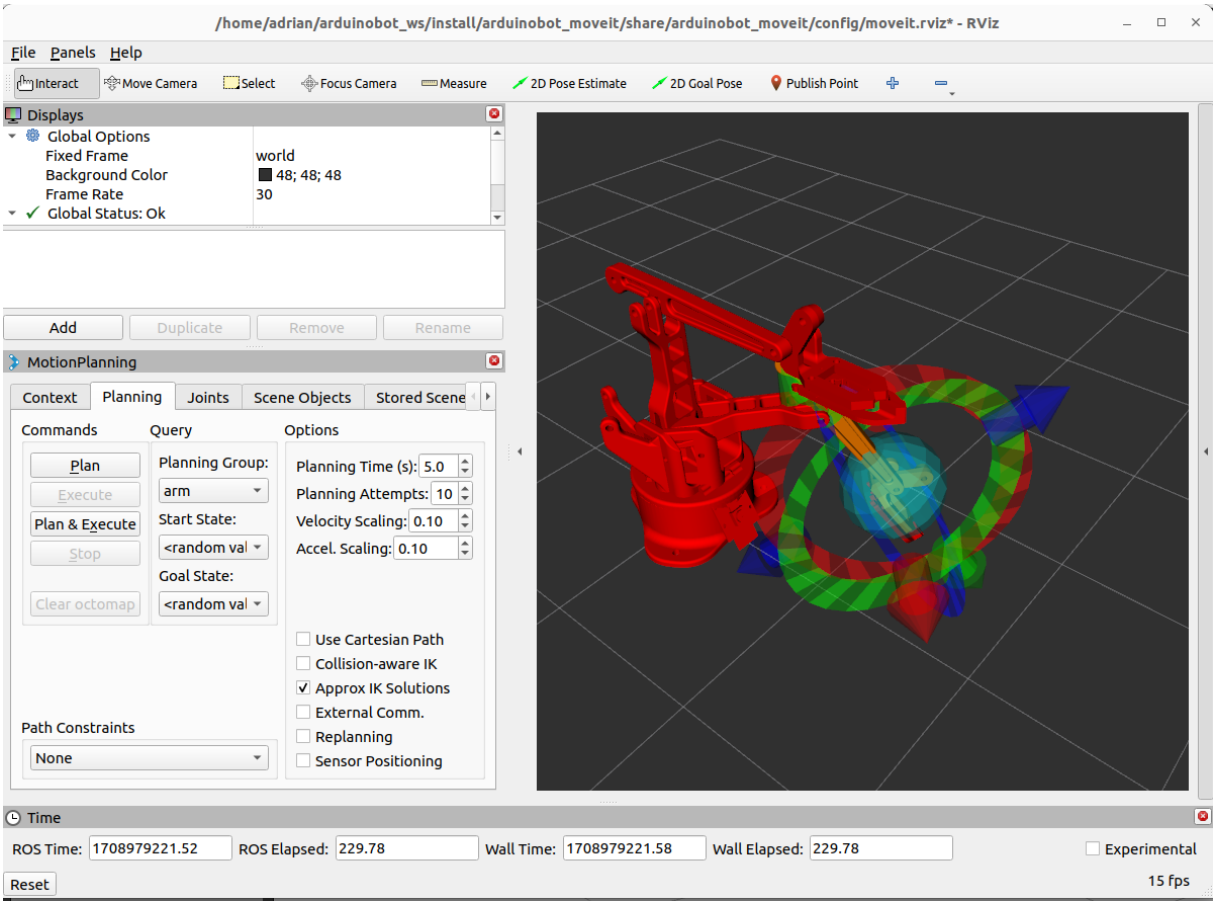


Figura 5.3: Fuente: Elaboración propia

Para escoger poses inválidas se hace uso de el posicionamiento manual del efector final en la herramienta. En la Fig. 5.4 se observa la forma en la que se coloca la tenaza. Su cuerpo se torna rojo por completo para indicar que la pose no puede ser alcanzada por el robot. Además, se observa que la configuración implica una colisión y de igual forma se restringe.

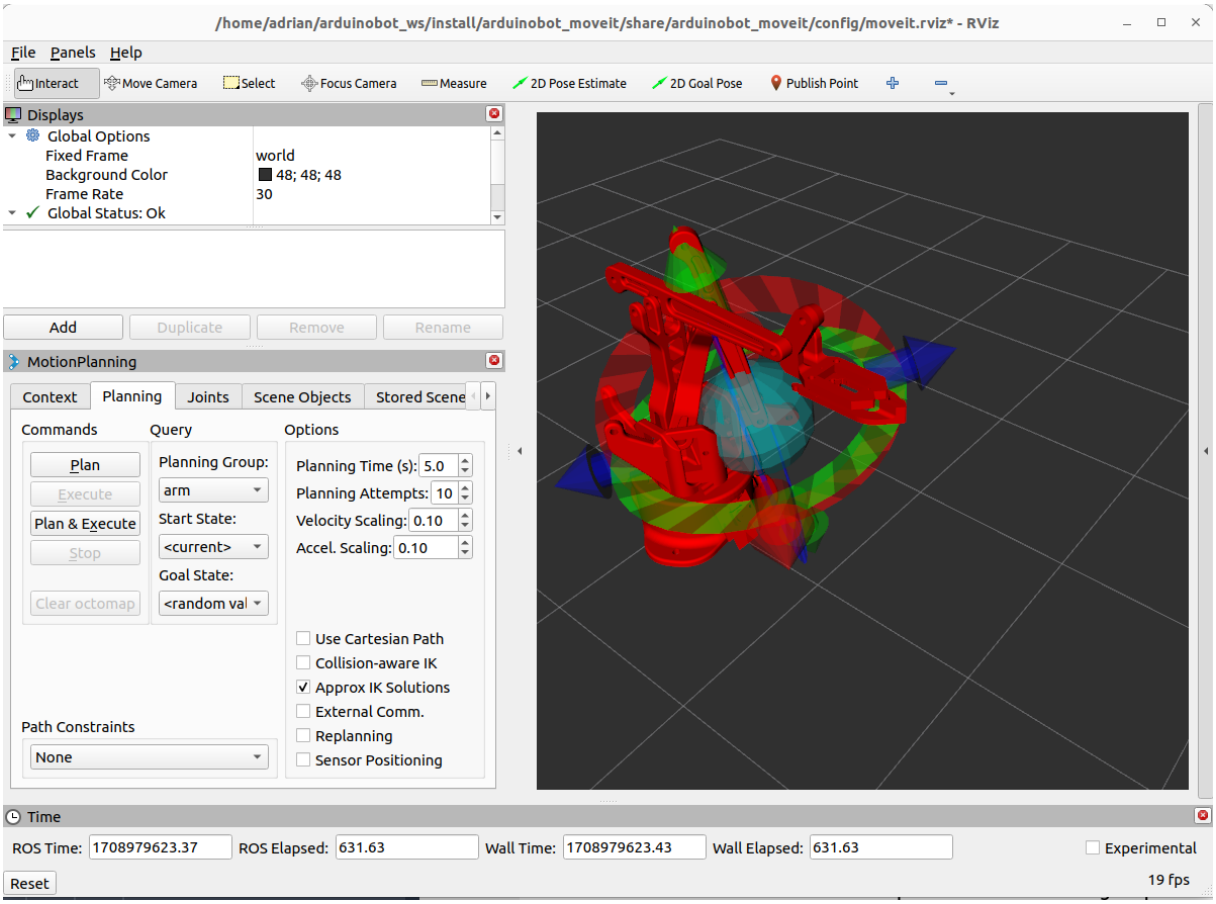


Figura 5.4: Fuente: Elaboración propia

Por medio del uso de la lógica descrita previamente se hacen 5 pruebas para poses válidas y inválidas. Las pruebas se ejecutan de forma intercalada. En la tabla 5.3 se muestra el resumen de las pruebas:

Tabla 5.3: Resultados experimentales de pruebas de colisión

No.	Tipo de pose	Resultado
1	Válida	Trayectoria ejecutada
2	No válida	Error de cinemática
3	Válida	Trayectoria ejecutada
4	No válida	Error de cinemática
5	Válida	Trayectoria ejecutada
6	No válida	Error de cinemática
7	Válida	Trayectoria ejecutada
8	No válida	Error de cinemática
9	Valida	Trayectoria ejecutada
10	No válida	Error de cinemática

5.3.2. Análisis de resultados

Los resultados de la simulación son excelentes, es decir, se evita al 100 % trayectorias que impliquen colisiones con el robot lo cual corresponde al valor ideal de las métricas objetivo especificadas para este proyecto. Algunas observaciones de este resultado son las siguientes:

1. El sistema retorna un mensaje de error indicando problemas de cinemática que impiden realizar la trayectoria cuando se asigna como objetivo una pose inválida en todas las pruebas realizadas.

2. Cuando se ingresan poses válidas el sistema indica que la cinemática se puede resolver y se procede a ejecutar el movimiento en el ambiente simulado.

3. El comportamiento simulado es consistente para todas las pruebas, no se presenta variación alguna en los resultados pero se tiene claro que en un sistema real habrán mayores diferencias de resultados.

```
1 [move_group-9] [INFO] [1709027062.562635070] [
    moveit_collision_detection_fcl.collision_common]: Collision checking
    is considered complete (collision was found and 0 contacts are stored
```

Listing 5.1: Ejemplo de mensaje de colisión detectada

Capítulo 6

Conclusiones

6.1. Conclusiones del rendimiento del proyecto

1. Se codifican 3 acciones atómicas de un robot de tipo manipulador que permiten simular el movimiento cinemático de trayectorias de sus articulaciones y efector final. Las acciones permiten llevar el robot a una posición de descanso, trabajo y objetivo. Todas las acciones sirven y su código puede ser editado por el cliente para generar objetivos distintos o agregar.

2. La mayor imprecisión del sistema es de 0.45 grados lo cual se encuentra por debajo del umbral ideal establecido en las especificaciones. La precisión alcanzada en la simulación excede las expectativas del cliente.

3. El sistema cuenta con capacidades de prevención de colisiones. Los resultados demuestran un porcentaje del 100 % lo cual cumple con los valores objetivo establecidos. No obstante, se atribuye este gran comportamiento a la naturaleza simulada del sistema.

4. Se documenta de forma descriptiva un flujo de diseño de manipuladores en *ROS* 2. Esta tecnología cumple con la necesidad del cliente de actualizar la base de datos de conocimiento en este kit de desarrollo de software y se evita que el proyecto quede obsoleto.

5. Se cumplen con todas las métricas establecidas para la simulación del robot con la arquitectura diseñada. Dicha arquitectura sirve para robots de tipo manipulador y se puede utilizar para ejecutar experimentos con las 3 acciones atómicas implementadas. Se concluye que la solución cumple con las 5 necesidades establecidas por el cliente por medio de la medición de 4 métricas relevantes al ambiente de robótica cognitiva simulado.

6. Se cumplen con los entregables y indicadores de los objetivos planteados para este proyecto. Se demuestra en forma de video cada una de las acciones atómicas y se

demuestra el correcto funcionamiento del ambiente simulado por medio de experimentos que retornan resultados que cumplen con los valores objetivo del proyecto.

6.2. Conclusiones de la economía del proyecto

1. El costo de esta solución es de 911118.4 colones. Este costo es menor al costo de $2000 especificado como valor objetivo usando un tipo de cambio de 508 colones. Se concluye que la solución se ajusta de forma exitosa al presupuesto del cliente.

2. El método de validación fue directamente dependiente de la cantidad de recurso económico disponible en el laboratorio. Debido a que no se cuenta con suficiente dinero para la reparación del modelo físico del robot *OSCAR*, se opta por utilizar una simulación que permite demostrar el concepto de un robot manipulador en *ROS* 2 a un costo accesible.

3. El recurso de tiempo del proyecto se administra para implementar un plan de remedio. Se invierten 8 semanas de las 16 semanas del tiempo del proyecto en buscar obtener fondos para reparar el robot *OSCAR*. No se obtienen fondos por lo cual se solicita una prórroga para la implementación de una alternativa de validación para el problema. Se debe planificar de forma cuidadosa las tareas para cumplir en el tiempo restante del proyecto.

4. El robot usado en este proyecto es más barato que otros modelos disponibles en el mercado y se puede construir con componentes que están en Costa Rica. Además, es 2.95 veces más barato que la reconstrucción de *OSCAR* lo cual permite al laboratorio tener una alternativa de robot para experimentos.

5. Desde un punto de vista social, este proyecto aporta a la base de datos del laboratorio *LIANA* dejando un beneficio tangible que puede servir para el desarrollo de futuras investigaciones en robótica cognitiva. Además implica la formación de un ingeniero para la sociedad costarricense.

6. Se resuelve la necesidad del cliente de la planificación de un plan de remedio para obtener fondos para *OSCAR* debido a que la documentación estrategia desarrollada para solicitar fondos de la *VIE* se entrega al laboratorio.

6.3. Recomendaciones

1. Se recomienda evitar el uso de maquinas virtuales para la simulación de *ROS* 2 debido a su requerimiento gráfico sobre el sistema [78]. En este proyecto, se comienza usando una máquina virtual porque el rendimiento de la simulación es pésimo y no permite experimentación. Se recomienda el uso de una instalación física del sistema operativo en una partición de memoria de la computadora.

6.4. Trabajo futuro

1. Para actualizar el modelo reconstruido de *OSCAR* se debe exportar su modelo reconstruido [4] a formato *URDF*. Para esto, una herramienta disponible en el mercado es SolidWorks ya que cuenta con un módulo de exportación URDF de ensambles [79]. Se recomienda que se preparen subensambles de cada articulación para lograr que sea compatible.

2. Para utilizar la arquitectura de robótica descrita en este trabajo con el modelo físico de *OSCAR* se debe ajustar todos los módulos descritos en el flujo de trabajo 4.1 para acomodar el hardware extra como los controladores, motores y articulaciones.

Bibliografía

[1] Instituto Tecnológico de Costa Rica. «Laboratorio de Inteligencia Artificial para las Ciencias Naturales (LIANA).» Consultado el 7 de septiembre del 2023. (2019), dirección: https://www.tec.ac.cr/unidades/laboratorio-inteligencia-artificial-ciencias-naturales-liana.

[2] E. F. Hernandez, «Diseño de entorno robótico como herramienta para el desarrollo de arquitecturas cognitivas,» Consultado el 9 de septiembre del 2023, Instituto Tecnológico de Costa Rica, 2021. dirección: https://repositoriotec.tec.ac.cr/handle/2238/13269.

[3] A. G. Vega, «Implementación de un sistema robótico para la realización de experimentos avanzados de robótica cognitiva,» Consultado el 9 de septiembre del 2023, Instituto Tecnológico de Costa Rica, 2022. dirección: https://github.com/ALEGOVE03/Construcci-n-OSCAR-.

[4] M. F. A. Soto, «Implementación de un sistema robótico para la realización de experimentos avanzados de robótica cognitiva,» Consultado el 9 de septiembre del 2023, Instituto Tecnológico de Costa Rica, 2023.

[5] J. Craig, *Robótica 3era edición* (Area ingenieria). Pearson Educación, 2006, Consultado el 14 de febrero del 2024, ISBN: 9789702607724.

[6] M. Mendoza. «Componentes que conforman un robot manipulador.» Consultado el 14 de febrero del 2024. (2019), dirección: http://ri.uaemex.mx/bitstream/handle/20.500.11799/108137/secme-33422_1.pdf?sequence=1&isAllowed=y#:~:text=Un%20manipulador%20rob%C3%B3tico%20consta%20de,de%20cada%20dos%20eslabones%20consecutivos.&text=El%20conjunto%20de%20eslabones%20y%20articulaciones%20se%20denomina%20cadena%20cinem%C3%A1tica..

[7] Mecharithm. «Other Explicit Representation for the Orientation in Robotics: Roll-Pitch-Yaw Angles.» Accedido el 27 de febrero del 2024. (2021), dirección: https://mecharithm.com/learning/lesson/explicit-representations-orientation-robotics-roll-pitch-yaw-angles-15.

[8] M. Rodríguez. «¿Qué son los grados de libertad (GDL) en un robot?» Accedido el 29 de febrero del 2024. (2023), dirección: https://www.inesem.es/revistadigital/gestion-integrada/diferencia-robotica-grados-libertad-movilidad-3/.

[9] D. Valencia. «Simulate a 6DoF Robotic Arm in Gazebo and ROS2.» Consultado el 8 de febrero del 2024. (2024), dirección: https://davidvalenciaredro.wixsite.com/my-site/services-7.

[10] H. Kutluca. «Robot Operating System 2 (ROS 2) Architecture.» Consultado el 8 de febrero del 2024. (2020), dirección: https://medium.com/software-architecture-foundations/robot-operating-system-2-ros-2-architecture-731ef1867776.

[11] Open Source Robotics Foundation. «Understanding ROS 2 Nodes.» Consultado el 9 de febrero del 2024. (2023), dirección: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html.

[12] D. Thomas, L. Moesenlechner y M. Beetz, «ROS 2.0: The Robot Operating System for the Next Decade,» *Proceedings of the ROSCon 2014 Conference*, 2014, Consultado el 10 de febrero del 2024. dirección: https://vimeo.com/106992622.

[13] Open Source Robotics Foundation. «Understanding ROS 2 Topics.» Consultado el 11 de febrero del 2024. (2023), dirección: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html.

[14] Open Source Robotics Foundation. «Understanding ROS 2 Services.» Consultado el 11 de febrero del 2024. (2023), dirección: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html.

[15] Open Source Robotics Foundation. «Understanding ROS 2 Actions.» Consultado el 12 de febrero del 2024. (2023), dirección: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html.

[16] Open Source Robotics Foundation. «Creating Your First ROS 2 Package.» Consultado el 12 de febrero del 2024. (2023), dirección: https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html.

[17] Open Source Robotics Foundation. «Building a Visual Robot Model with URDF from Scratch.» Consultado el 12 de febrero del 2024. (2023), dirección: https://docs.ros.org/en/humble/Tutorials/Intermediate/URDF/Building-a-Visual-Robot-Model-with-URDF-from-Scratch.html.

[18] ultroninverse. «How to Create the URDF in a Correct Way.» Consultado el 13 de febrero del 2024. (2022), dirección: https://medium.com/@ultroninverse/how-to-create-the-urdf-in-a-correct-way-4641a66d1a5.

[19] E. Demaitre. «MoveIt 2 enables realtime robot arm control with ROS 2.» Accedido el 27 de febrero del 2024, The Robot Report. (2020), dirección: https://www.therobotreport.com/moveit-2-enables-realtime-robot-arm-control-ros2/.

[20] «Quaternion fundamentals.» Accedido el 27 de febrero del 2024, ROS 2 Documentation: Humble. (2024), dirección: https://docs.ros.org/en/humble/Tutorials/Intermediate/Tf2/Quaternion-Fundamentals.html.

[21] L. Poubel. «Open Source Robotics: Getting Started with Gazebo and ROS 2.» Consultado el 15 de febrero del 2024. (2019), dirección: https://www.infoq.com/articles/ros-2-gazebo-tutorial/.

[22] N. Koenig y A. Howard, «Design and use paradigms for Gazebo, an open-source multi-robot simulator,» en *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, Consultado el 12 de febrero del 2024, vol. 3, 2004, 2149-2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[23] D. L. AI y Robotics. «Robotics with ROS 2.» Consultado el 13 de febrero del 2024. (2022), dirección: https://robotics.snowcron.com/robotics_ros2/robotics_ros2.htm.

[24] Robotogeddon. «Gazebo Simulator: GUI Explained - Part I.» Consultado el 13 de febrero del 2024. (2021), dirección: https://www.youtube.com/watch?v=HDboF7Itra8.

[25] Gazebo Tutorials. «Sensors | Gazebo Docs.» Consultado el 13 de febrero del 2024. (2023), dirección: https://gazebosim.org/docs/latest/sensors.

[26] U. P. de Catalunya. «Gazebo Sensors Tutorial.» Consultado el 13 de febrero del 2024. (2022), dirección: https://sir.upc.edu/projects/rostutorials/8-gazebo_sensors_tutorial/index.html.

[27] A. W. Services. «What is an API? - Application Programming Interface Explained.» Accedido el 27 de febrero del 2024. (2024), dirección: https://aws.amazon.com/what-is/api/.

[28] S. Soni. «What is Ngrok and How Does It Work?» Accedido el 27 de febrero del 2024, Requestly. (2023), dirección: https://requestly.io/blog/what-is-ngrok-and-how-does-it-work/.

[29] M. Bizzaco. «What is Amazon's Alexa, and What Can It Do?» Accedido el 27 de febrero del 2024, Digital Trends. (2023), dirección: https://www.digitaltrends.com/home/what-is-amazons-alexa-and-what-can-it-do/.

[30] S. D. E. Karl T. Ulrich, *Diseño y desarrollo de productos*, 5ta. McGraw Hill Education, 2020, Consultado el 12 de septiembre del 2023.

[31] EdrawSoft. «Método 6M: Cómo utilizar el diagrama de Ishikawa en Six Sigma.» Consultado el 7 de septiembre de 2023. (Fecha de acceso: 2023-09-07), dirección: https://www.edrawsoft.com/es/6m-method.html.

[32] D. C. J. Marcos García Bartolomé Jose Alvarez Ontivero. «Robotrónica: Aplicaciones de la Robótica.» Consultado el 23 de septiembre del 2023. (2003), dirección: https://upcommons.upc.edu/bitstream/handle/2117/186000/40449-3452.pdf.

[33] M. R. Miller y R. Miller, «Arm,» en, en *Robots and Robotics: Principles, Systems, and Industrial Applications*, First edition. New York: McGraw-Hill Education, 2017, Consultado el 23 de septiembre del 2023, ISBN: 9781259859786. dirección: https://www.accessengineeringlibrary.com/content/book/9781259859786/toc-chapter/chapter2/section/section11.

[34] Open Source Robotics Foundation. «ROS - Robot Operating System.» Consultado el 23 de septiembre del 2023. (2023), dirección: https://www.ros.org/.

[35] F. A. Research. «PyRobot - An Open-source AI Research Framework for Robotics.» Consultado el 23 de septiembre del 2023. (2023), dirección: https://pyrobot.org/.

[36] A. M. Technologies. «PLC Controlled Robots.» Consultado el 23 de septiembre del 2023. (2020), dirección: https://blog.appliedmfg.com/blog/plc-controlled-robots.

[37] Glowbuzzer. «How it works.» Consultado el 23 de septiembre del 2023. (2023), dirección: https://www.glowbuzzer.com/how-it-works/overview.

[38] I. Technologies. «What is Human-Machine Interaction?» Consultado el 23 de septiembre del 2023. (2023), dirección: https://www.infineon.com/cms/en/discoveries/human-machine-interaction/.

[39] M. Driscoll. «PySimpleGUI: The Simple Way to Create a GUI With Python.» Consultado el 28 de septiembre del 2023. (2022), dirección: https://realpython.com/pysimplegui-python/#conclusion.

[40] V. Khandelwal. «A One-Stop Solution for Creating a C GUI.» Consultado el 30 de septiembre del 2023. (2023), dirección: https://www.simplilearn.com/tutorials/c-sharp-tutorial/c-sharp-gui.

[41] Gazebo. «Features.» Consultado el 30 de septiembre del 2023. (2023), dirección: https://gazebosim.org/features.

[42] RoboDK. «Simulate and Program your Robot in 5 Easy Steps.» Consultado el 3 de octubre del 2023. (2023), dirección: https://robodk.com/simulation.

[43] Picknik. «MoveIt Open Source.» Consultado el 4 de octubre del 2023. (2023), dirección: https://picknik.ai.

[44] Dassault. «Delmia.» Consultado el 8 de octubre del 2023. (2023), dirección: https://www.3ds.com/products/delmia/industrial-engineering/robotics.

[45] A. Gomariz. «Los robots podrían replicar la percepción sensorial humana.» Accedido el 27 de febrero del 2024. (2019), dirección: https://blogthinkbig.com/robots-percepcion-sensorial-humana.

[46] Amazon. «Custom Skill Developer Reference.» Accedido el 25 de febrero del 2024. (2024), dirección: https://developer.amazon.com/en-US/docs/alexa/reference/custom-skill-developer-reference.html.

[47] P. Gazarov. «What is an API? In English, please.» Accedido el 27 de febrero del 2024. (2017), dirección: https://www.freecodecamp.org/news/what-is-an-api-in-english-please-b880a3214a82/.

[48] OpenCV. «Services.» Consultado el 15 de octubre del 2023. (2023), dirección: https://www.opencv.ai.

[49] Scikit. «Documentation.» Consultado el 16 de octubre del 2023. (2023), dirección: https://scikit-image.org/docs/stable.

[50] J. Clark. «Pillow 10.2.0.» Consultado el 20 de octubre del 2023. (2023), dirección: https://pypi.org/project/pillow/.

[51] R. E. Walpole y Myers, *Probabilidad y Estadística para Ingeniería y Ciencias*. 2017, Accedido el 29 de febrero del 2024. dirección: https://bibliotecavirtualaserena.files.wordpress.com/2017/05/libro_probabilidad-y-estadistica-para-ingenierc3ada-y-ciencias-ronald-e-walpole-mayers.pdf.

[52] L. West. «Pros and Cons of PLA: Corn-Based Plastic.» Consultado el 25 de octubre de 2023. (2020), dirección: https://www.treehugger.com/pros-cons-corn-based-plastic-pla-1203953.

[53] QuickBooks. «¿Qué es la depreciación contable?» Consultado el 5 de febrero del 2024. (2024), dirección: https://quickbooks.intuit.com/global/resources/es/contabilidad/que-es-depreciacion-en-contabilidad/.

[54] MicroJPM. «Mini Servo TowerPro SG90 con Accesorios.» Accedido el 29 de febrero del 2024. (2024), dirección: https://www.microjpm.com/products/mini-servo-towerpro-sg90-con-accesorios/.

[55] M. JPM. «Arduino UNO R3.» Accedido el 29 de febrero del 2024. (2024), dirección: https://www.microjpm.com/products/arduino-uno-r3-version-generica/.

[56] M. JPM. «M3*6mm Tornillo Philips.» Accedido el 29 de febrero del 2024. (2024), dirección: https://www.microjpm.com/products/m3-x-8mm-tornillo-philips/.

[57] A. Brandi. «Robotics and ROS 2: Learn by Doing - Manipulators.» Consultado el 19 de febrero del 2024. (2024), dirección: https://github.com/AntoBrandi/Robotics-and-ROS-2-Learn-by-Doing-Manipulators.

[58] RobotShop. «Lynxmotion SES-V2 Robotic Arm (3 DoF) w/ Smart Servos Kit.» Accedido el 29 de febrero del 2024. (2024), dirección: https://www.robotshop.com/products/lynxmotion-lss-3-dof-robotic-arm-kit.

[59] «Robotic Arm Mechanical Manipulator with Teaching Controller.» Accedido el 29 de febrero del 2024, Amazon. (2024), dirección: https://www.amazon.com/Robotic-Mechanical-Manipulator-Teaching-Controller/dp/B0CKT2TQLK?th=1.

[60] C. Electronics. «3DOF Robot Arm Kit (DIY, Servos included).» Accedido el 29 de febrero del 2024. (2024), dirección: https://core-electronics.com.au/3dof-robot-arm-kit-diy.html.

[61] «3-Axis Industrial Robotic Mechanical Arm 3DOF Load Capacity 500g w/ Controller.» Accedido el 29 de febrero del 2024. (2024), dirección: https://www.ebay.com/itm/165655628251.

[62] B. Vinod, B. Bindu, G. Koushik Karan, V. E. Jayanth Akash y S. Dinesh Kumar, «Design and implementation of the 6-DoF robotic manipulator using robot operating system,» *International Journal of Nonlinear Analysis and Applications*, vol. 12, n.º Special Issue, págs. 1753-1760, 2021, Consultado el 19 de febrero del 2024, ISSN: 2008-6822. DOI: 10.22075/ijnaa.2021.5882. eprint: https://ijnaa.semnan.ac.ir/article_5882_e6d960e4077d196015905cd8e102eadd.pdf. dirección: https://ijnaa.semnan.ac.ir/article_5882.html.

[63] W. Karonji. «Arduino: Mastering Serial Communication.» Accedido el 10 de febrero del 2024, MakeUseOf. (2023), dirección: https://www.makeuseof.com/arduino-mastering-serial-communication/.

[64] «Smraza 10 Pcs SG90 9G Micro Servo Motor Kit for RC Robot Arm Helicopter Airplane Remote Control.» Accedido el 20 de febrero del 2024, Amazon. (2024), dirección: https://www.amazon.com/Smraza-Geared-Control-Arduino-Project/dp/B07L2SF3R4/.

[65] «Download Ubuntu.» Consultado el 19 de febrero del 2024, Canonical Ltd. (2024), dirección: https://ubuntu.com/download.

[66] «ROS Humble Hawksbill Installation.» Consultado el 19 de febrero del 2024, Open Robotics. (2024), dirección: https://docs.ros.org/en/humble/Installation.html.

[67] «Gazebo Documentation.» Consultado el 19 de febrero del 2024, Open Robotics. (2024), dirección: https://gazebosim.org/docs.

[68] A. Bacchus. «How to Install VS Code on Ubuntu.» Consultado el 19 de febrero del 2024, XDA Developers. (2023), dirección: https://www.xda-developers.com/how-to-install-vs-code-on-ubuntu/.

[69] A. Sears. «Naming and Organizing Packages in Large ROS 2 Projects.» Accedido el 10 de febrero del 2024, Automatic Addison. (2021), dirección: https://automaticaddison.com/naming-and-organizing-packages-in-large-ros-2-projects/.

[70] J. Juviler. «What is an XML File.» Accedido el 20 de febrero del 2024, HubSpot. (), dirección: https://blog.hubspot.com/website/what-is-xml-file.

[71] «joint_trajectory_controller — ROS2_Control: Humble.» Accedido el 24 de febrero del 2024, Open Robotics. (2024), dirección: https://control.ros.org/humble/doc/ros2_controllers/joint_trajectory_controller/doc/userdoc.html.

[72] «SimpleTransmission Class Reference.» Accedido el 24 de febrero del 2024, Open Robotics. (2024), dirección: http://docs.ros.org/en/melodic/api/transmission_interface/html/c++/classtransmission__interface_1_1SimpleTransmission.html.

[73] O. Robotics. «gazebo_ros2_control Documentation.» Accedido el 24 de febrero del 2024. (2024), dirección: https://control.ros.org/master/doc/gazebo_ros2_control/doc/index.html.

[74] J. L. Millán. «How to Use ROS2 Parameters.» Accedido el 24 de febrero del 2024. (2022), dirección: https://foxglove.dev/blog/how-to-use-ros2-parameters.

[75] R. Alves. «Learn TF2 ROS2: Concepts in Practice.» Accedido el 25 de febrero del 2024. (2022), dirección: https://www.theconstructsim.com/learn-tf2-ros2-concepts-in-practice/.

[76] P. Robotics. «MoveIt Configuration Tutorial.» Accedido el 25 de febrero del 2024. (2024), dirección: https://moveit.picknik.ai/main/doc/how_to_guides/moveit_configuration/moveit_configuration_tutorial.html.

[77] M. Day. «ROS Topics RQTGraph.» Accedido el 25 de febrero del 2024. (2022), dirección: https://www.keypuncher.net/blog/ros-topics-amp-rqtgraph.

[78] N. Koenig, *Run Gazebo in VirtualBox*, Accedido el 26 de febrero del 2024, Gazebo, 2012. dirección: https://answers.gazebosim.org//question/25/run-gazebo-in-virtualbox/.

[79] R. W. Contributors. «sw$_u rdf_e xporter$.» Accedido el 26 de febrero del 2024. (2024), dirección: http://wiki.ros.org/sw_urdf_exporter.

Apéndice A

Apendice

A.1. Repositorio del proyecto

A continuación se provee un link al repositorio del proyecto:

Link de repositorio

A.2. Resultados experimentales

En base a los siguientes datos, se calculan los resultados experimentales de la precisión de rotación de las articulaciones.

Tabla A.1: Datos de experimento 2 en posición objetivo

No.	roll (rad)	pitch (rad)	yaw (rad)	gripper left	gripper right
1	-1.13	-0.59	-0.07	0	0
2	-1.12	-0.59	-0.07	0	0
3	-1.12	-0.6	-0.07	0	0
4	-1.14	-0.6	-0.07	0	0
5	-1.14	-0,6	-0.07	0	0
6	-1.14	-0.6	-0.07	0	0
7	-1.13	-0.6	-0.07	0	0
8	-1.14	-0.58	-0.07	0	0
9	-1.14	-0.6	-0.07	0	0
10	-1.14	-0.6	-0.07	0	0
Promedio	-1.134	-0.596	-0.07	0	0
Objetivo	-1.14	-0.6	-0.07	0	0
Diferencia	0.006	0.004	0	0	0
Desv. Estandar	0.008	0.007	0	0	0

Tabla A.2: Datos de experimento 2 en posición de descanso

No.	roll (rad)	pitch (rad)	yaw (rad)	gripper left	gripper right
1	-1.56	0	-0.9	0	0
2	-1.56	0	-0.89	0	0
3	-1.56	0	-0.89	0	0
4	-1.56	0	-0.89	0	0
5	-1.57	0	-0.89	0	0
6	-1.57	0	-0.9	0	0
7	-1.56	0	-0.89	0	0
8	-1.57	0	-0.89	0	0
9	-1.56	0	-0.89	0	0
10	-1.56	0	-0.89	0	0
Promedio	-1.563	0	-0.892	0	0
Objetivo	-1.57	0	-0.9	0	0
Diferencia	0.007	0	0.008	0	0
Desv. Estándar	0.005	0	0.004	0	0

Tabla A.3: Datos de experimento 2 en posición de trabajo

No.	roll (rad)	pitch (rad)	yaw (rad)	gripper left	gripper right
1	0	0	0	-0.69	0.69
2	0.01	0	0	-0.69	0.7
3	0	0	0	-0.7	0.7
4	0	0	0.01	-0,7	0.7
5	0	0	0	-0,7	0.7
6	0	0	0	-0,7	0.7
7	0	0	0	-0,7	0.7
8	0.01	0	0	-0,7	0.7
9	0.01	0	0	-0,7	0.7
10	0	0	0	-0,7	0.7
Promedio	0.003	0	0.001	-0.69	0.69
Objetivo	0	0	0	-0.7	0.7
Diferencia	0.003	0.000	0.001	0.007	0.001
Desv. Estándar	0.005	0	0.003	0.006	0.003