Escuela Superior Politécnica del Litoral

Facultad de Ingeniería en Mecánica y Ciencias de la Producción

Diseño e implementación de un sistema Multi-Robot de código abierto para ambientes colaborativos en ROS2.

Proyecto Integrador

Previo la obtención del Título de:

Ingenieros en Mecatrónica

Presentado por:
Anthonny Ernesto Piguave Toala
Diego Sebastian Ronquillo Manosalvas

Guayaquil - Ecuador
Año: 2023

Dedicatoria

El presente proyecto lo dedico a mis padres, hermano, familia, amigos, compañeros de carrera, profesores de la universidad y profesores de mi colegio quienes me apoyaron y que gracias a ellos he llegado a cumplir un nuevo reto.

- **Anthonny Piguave**

Dedicatoria

El presente trabajo se lo dedico a mis padres, hermanos, abuelos y amigos; que siempre me han brindado apoyo y amor.

- **Diego Ronquillo**

Agradecimientos

Mi más sincero agradecimiento a la universidad, facultad y docentes de la carrera de Ingeniería en Mecatrónica, por su trabajo y dedicación.

Al laboratorio RAMEL, por darnos la oportunidad de trabajar en conjunto para realizar el presente trabajo.

A mi compañero de tesis, Diego Ronquillo, con quien compartí ideas, conocimientos y experiencias para completar este reto.

- **Anthonny Piguave**

Agradecimientos

Agradezco a mi familia y amistades por siempre estar.

A mi compañero, Anthonny Piguave, por haber sido un gran complemento para el desarrollo del proyecto integrador y haber dado siempre su mayor esfuerzo.

A RAMEL, por haber brindado instalaciones y recursos necesarios para el desarrollo de las investigaciones del documento.

Finalmente, a la ESPOL por la educación de calidad brindada.

- **Diego Ronquillo**

Declaración Expresa

"Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; Anthonny Piguave y Diego Ronquillo damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Anthonny Piguave Toala

Diego Ronquillo Manosalvas

Evaluadores

Firmado electrónicamente por:
MARCELO RODOLFO
FAJARDO PRUNA

Marcelo Fajardo Pruna, Ph.D.

Profesor de Materia

Firmado electrónicamente por:
FRANCISCO XAVIER
YUMBLA AREVALO

Francisco Yumbla Arevalo, Ph.D.

Tutor de proyecto

Resumen

En la actualidad, a pesar del auge de la robótica y la automatización en aplicaciones industriales, la adopción generalizada de la robótica colaborativa sigue siendo limitada debido a la falta de interoperabilidad entre robots y la baja adaptabilidad de los sistemas existentes. Solucionar esta problemática supondría un avance importante en la robótica y automatización industrial. Bajo ese contexto, en el presente proyecto se desarrolló un FrameWork MultiRobot de código abierto basado en ROS2 para la comunicación y coordinación efectiva de sistemas mecatrónicos y sensores en entornos colaborativos cerrados. Se realizó un diseño de software y control basado en simulación utilizando la herramienta de Gazebo. Se obtuvo una arquitectura centralizada con un módulo de navegación autónoma para la planificación y seguimiento de rutas de los robots, un módulo de visión por computadora para la localización y manejo de incertidumbres, y un módulo controlador de tareas para la asignación de misiones de movilización de objetos. En conclusión, utilizando ROS2 para lograr la comunicación y coordinación eficaz de diversos sistemas mecatrónicos se logra obtener una solución robusta, flexible y escalable; parámetros claves en los procesos industriales.

Palabras Clave: robótica colaborativa, interoperabilidad, visión artificial, navegación autónoma, escalabilidad.

Abstract

Nowadays, despite the rise of robotics and automation in industrial applications, the widespread adoption of collaborative robotics remains limited due to the lack of interoperability between robots and the low adaptability of existing systems. Solving this problem would mean an important advance in robotics and industrial automation. Under this context, in the present project an open source MultiRobot Framework based on ROS2 was developed for the effective communication and coordination of mechatronic systems and sensors in closed collaborative environments. A simulation-based control and software design was performed using the Gazebo tool. A centralized architecture was obtained with an autonomous navigation module for the planning and monitoring of robot routes, a computer vision module for the location and management of uncertainties, and a task controller module for the assignment of mobilization missions. objects. In conclusion, using ROS2 to achieve effective communication and coordination of various mechatronic systems, results in a robust, flexible, and scalable solution, key parameters in industrial processes.

Keywords: collaborative robotics, interoperability, artificial vision, autonomous navigation, scalability.

Índice general

Abreviaturas

ESPOL	Escuela Superior Politécnica del Litoral
RAMEL	Robotics, Automation & Mechatronics Engineering Laboratory
SMR	Sistema Multi Robot
UGV	Unmanned Ground Vehicle
UAV	Unmanned Aerial Vehicle
LiDAR	Light Detection and Ranging
SLAM	Simultaneous Localization and Mapping
ISO	International Organization for Standardization
BT	Behavior Tree (Árbol de comportamiento)
Tf	Marco de coordenada

Simbología

m	Metro
mm	Milímetro
°	Grado
Hz	Hercio

Índice de figuras

Índice de tablas

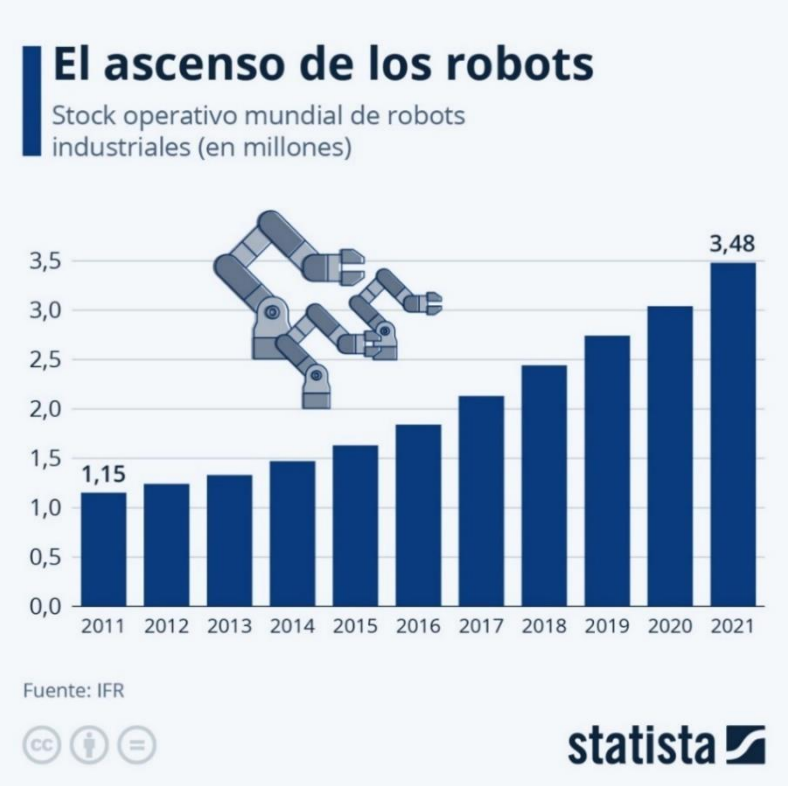
Capítulo 1

1.1 Introducción

En años recientes, se han presenciado avances significativos en el campo de la robótica y la automatización. Los robots se han convertido en colaboradores flexibles en una amplia gama de entornos, con una gran adaptabilidad. En aplicaciones industriales, servicios de medicina, agricultura de precisión y muchas otras áreas; los robots colaboran activamente con los humanos para desarrollar misiones requeridas. En el ámbito de la robótica industrial, resulta evidente el crecimiento sostenido de la cantidad de robots industriales en operación año tras año, como se muestra en la Figura 1.1. La operación efectiva de sistemas mecatrónicos se está volviendo cada vez más importante debido a que tiene el potencial para dar solución a problemas complejos y mejorar la eficiencia de las aplicaciones antes mencionadas.

Figura 1.1

Crecimiento de robótica industrial



Fuente: M. Mena Roa [1]**.**

Una de las principales bases de la robótica colaborativa, siendo de los avances más significativos, es Robot Operating System 2. ROS2 se trata de una colección de herramientas y frameworks de código abierto para el desarrollo de aplicaciones robóticas. Su notable capacidad para la comunicación y coordinación entre diferentes robots o los componentes de un mismo robot, han provocado que sea adoptado por la comunidad de investigadores y desarrolladores de robots.

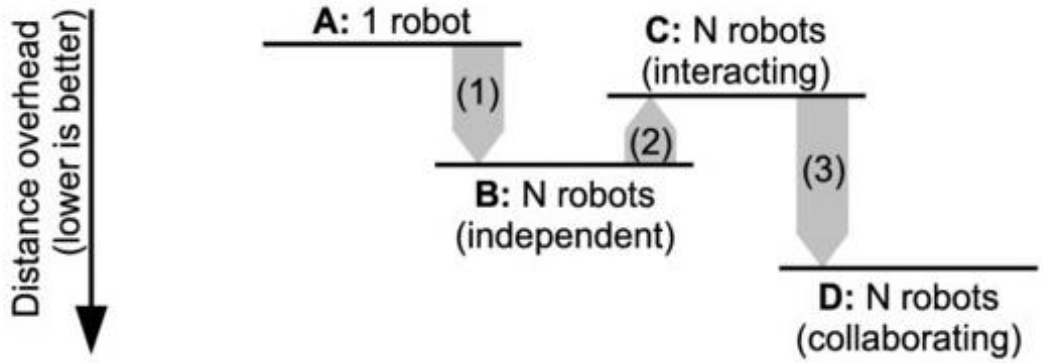
Se ha comprobado que la implementación de un Sistema Multi-Robot (SMR) es más rentable que la construcción de un robot con múltiples funcionalidades necesarias [2]. Integrar varios robots permite al sistema tener una mayor robustez, una capacidad más amplia de cobertura y mejorar su eficiencia. Por lo tanto, adoptar la colaboración de varios robots se está convirtiendo en una prioridad para las industrias y empresas. Desde la participación en procesos industriales, equipos de búsqueda o servicios de entrega en hospitales, restaurantes y galpones; las posibilidades de aplicaciones de sistemas conformados por múltiples robots son abundantes. Incluso misiones o tareas que en la actualidad pueden ser ejecutadas por un solo robot con varias funcionalidades se pueden beneficiar por un SMR, dado que la robustez y confiabilidad puede aumentar al combinar varios robots con una menor robustez y confiabilidad [3].

En la Figura 1.2 se muestran los resultados de una comparación entre distintos sistemas robóticos en una aplicación de localización de una fuente de olor. El parámetro de rendimiento es la distancia recorrida hasta hallar la fuente; a menor distancia, mejor rendimiento. Se puede apreciar que el sistema de un robot es el que peor rendimiento tiene. Si se realiza pruebas con N robots independientes en ambientes separados, el rendimiento aumenta debido al factor de la aleatoriedad. Sin embargo, este caso no es práctico. Para un sistema de N robots interactuando en un mismo ambiente, se ve una pequeña pérdida de rendimiento debido a las obstrucciones físicas entre robots. Finalmente, si se tiene un sistema de N robots colaborando,

la mejora de rendimiento es considerable, ya que entre N robots comunicándose y colaborando lograrán encontrar la fuente de olor más rápido. En consecuencia, los SMR es un área emergente con grandes beneficios que se encuentra en vigente investigación.

Figura 1.2

Comparación entre diferentes sistemas robóticos



Fuente: T. Lochmatter y A. Martinoli [4]**.**

1.2 Descripción del problema

Actualmente, los diseñadores de robots ofrecen soluciones específicas para tareas definidas y simples, lo que da como resultado sistemas robóticos fragmentados y una interoperabilidad deficiente. Cada desarrollador deja la responsabilidad al usuario de integrar y ajustar estos sistemas para lograr objetivos más complejos. Esta falta de interoperabilidad crea ineficiencias, dificulta la colaboración y limita el potencial de los sistemas robóticos. El usuario, al encontrarse con la carga laboriosa de acoplar diferentes robots, opta por trabajar con los robots de manera aislada, desperdiciando la capacidad del grupo de robots que posee. Además, el proceso de integrar robots e implementar tareas coordinadas está expuesta a errores; provocando mayor rechazo por parte de los usuarios.

El despliegue, seguimiento, control y recogida de un grupo de robots representan un desafío más grande que el de un robot individual. Además, el operador se enfrenta a una mayor dificultad para el control del sistema [5]. Por lo tanto, abordar este problema requiere proveer al usuario una ayuda o guía que cumpla con las siguientes exigencias: intercambio constante

en tiempo real de información importante entre los diferentes robots, coordinación eficiente para tareas colaborativas, habilidad para adaptarse a cambios en entornos dinámicos y gestión eficiente de recursos computacionales.

De igual importancia, hay que considerar las restricciones más comunes de la problemática. Entre las principales restricciones, hay que garantizar la integridad física de las personas que se encuentren en el ambiente colaborativo y la seguridad del entorno. Además, se debe tomar en cuenta las limitaciones propias de los robots que el usuario pueda poseer, tales como: energía, potencia de procesamiento y almacenamiento. También, se deben minimizar los costos adheridos a la implementación del sistema. La escalabilidad del sistema, para manejar un mayor número de robots y sensores, es la principal variable de interés de la problemática. Asimismo, se debe tomar en cuenta el tiempo de respuesta, la eficiencia en el cumplimiento de misiones, la robustez y la calidad de la colaboración.

La idea de este proyecto integrador surge de un problema en RAMEL, el cual es una instalación ubicada en ESPOL dedicadas a la investigación y educación en el campo de la robótica. RAMEL cuenta con diversos tipos de robots, incluyendo UGV y UAV, además de varios sensores y cámaras de alta resolución. El laboratorio requiere el funcionamiento de 2 TurtleBot3 y un Create3 Robot en su ambiente colaborativo cerrado, en el cual trabajan diariamente personas. Las tareas que deben realizar los robots son de recogida o entrega de objetos.

1.3 Justificación del problema

Solventar el problema descrito en la anterior sección requiere el diseño y desarrollo de un framework para SMR código abierto que permita una comunicación y coordinación efectiva entre diferentes sistemas mecatrónicos. El sistema debe ser asequible, adaptable y seguro para su implementación en entornos de colaboración humano-robot. Es importante resolver este problema ya que las dificultades para implementar un SMR en industrias y empresas restringen la adopción en entornos reales, siendo un obstáculo para que la robótica colaborativa progrese. Además, las empresas dueñas de robots no están sacando el máximo provecho al potencial y capacidades de estos.

La implementación de un SMR en un entorno colaborativo permite mejorar la eficiencia en el cumplimiento de misiones requeridas. También, posibilita la asignación de misiones con una mayor complejidad, ya que se cuenta con un mayor número de recursos. Otra ventaja es su robustez. Si uno de los robots o sensores llega a tener un fallo, se puede culminar con la realización de la tarea gracias al resto de robots o sensores. Esta flexibilidad es un factor esencial y requerido en las industrias. Finalmente, el intercambio de información constante entre los sistemas y equipos de un SMR permite una cobertura extendida. Dicha funcionalidad es relevante en aplicaciones de exploración y vigilancia de grandes espacios. Las empresas o industrias cuentan con grandes instalaciones, por lo que para realizar una cobertura total se beneficiarían ampliamente de este sistema planteado, mejorando su productividad y aprovechamiento.

En resumen, la implementación de un SMR permite aprovechar el potencial de los sistemas mecatrónicos pertenecientes a un usuario. Tanto la eficiencia, como la robustez y la cobertura de la misión asignada serán mejoradas. El diseño de un framework para SMR utilizando ROS2 es un desafío relevante en el campo de la robótica, puesto a que involucra localización, comunicación y navegación autónoma de varios robots.

1.4 Objetivos

1.4.1 Objetivo general

Diseñar un FrameWork MultiRobot Open Source mediante la comunicación y coordinación efectiva de diferentes sistemas mecatrónicos y sensores utilizando ROS2, para la implementación en un ambiente cerrado colaborativo.

1.4.2 Objetivos específicos

1. Diseñar una arquitectura de software multirobot en ROS2 para la configuración y control de varios robots logrando sinergia con la comunicación y coordinación efectiva entre los agentes.
2. Desarrollar un ambiente simulado utilizando Gazebo para validar la interacción de múltiples robots.
3. Implementar la detección de marcadores Aruco utilizando una cámara, para permitir la localización y orientación precisa de los robots y los espacios de trabajo en el entorno de simulación.
4. Aplicar técnicas de SLAM utilizando sensores LiDAR para realizar el mapeo y navegación autónoma en un ambiente cerrado virtual.
5. Elaborar un algoritmo de control para la ejecución de misiones de movilización de objetos que los robots deban ejecutar coordinadamente, realizando una asignación de tareas en función de la posición de los agentes.

1.5 Marco teórico

En la presente sección se procede a describir fundamentos técnicos relacionados con la problemática descrita anteriormente, además de explorar investigaciones, proyectos y soluciones relacionadas con la solución propuesta.

1.5.1 Industria 4.0

Se entiende por industria 4.0 al "desarrollo de industrias inteligentes con una mayor productividad de productos de alta calidad que satisfagan las expectativas del cliente. Esto requiere flexibilidad, automatización e interconexión." [6, p. 206]. En [7] se indica que muchas empresas a nivel mundial, en especial las dedicadas a la manufactura, han procedido a reconfigurar sus sistemas de producción, a partir de este nuevo concepto, con el fin de implementar máquinas, productos e infraestructura que colaboren entre sí, compartiendo datos e información con el fin de dotar al sistema de flexibilidad y robustez, aumentar la productividad mediante la automatización del proceso, y obtener un sistema con equipos y procesos interconectados y en constante comunicación no solo con otras máquinas, sino también con los operadores.

Según Sachon [7], para que una empresa pueda implementar el modelo de industria 4.0 debe considerar 5 pilares fundamentales: La generación y captura de datos, el análisis de los datos, la interacción hombre-máquina, la producción flexible y la propiedad intelectual. El auge de este concepto de industria ha ocasionado la denominada "cuarta revolución industrial", donde las empresas buscan incorporar nuevas tecnologías con el fin de mejorar su productividad, mientras que los centros de investigación y desarrollo conducen sus trabajos en el campo de la robótica, big data e inteligencia artificial para ofrecer soluciones al sector industrial, buscando facilitar la implementación de este modelo de industria en función de los pilares descritos.

De esta manera, tecnologías como el internet que facilita la comunicación y recolección de datos, realidad aumentada y gemelos digitales que facilitan la interacción entre el operador y la máquina o proceso, y la integración de robots colaborativos orientados a interactuar con las personas mediante el uso de sensores en sus espacios de trabajo siguiendo las normas de seguridad requeridas, reflejan los esfuerzos por promover la digitalización en las empresas. No obstante, dado el último pilar, es importante prestar la debida atención al manejo de los datos e información en la red, tanto por protección de la empresa como de información personal que puede llegar a ser capturada y utilizada por terceros si no se implementa un buen sistema de seguridad.

1.5.2 Robots y Sistemas Multi-Robots

En pro de la industria 4.0, el desarrollo de nuevas tecnologías debe estar orientado a cumplir con cada uno de los pilares, de tal manera, un robot, por ejemplo, no solo debe ser capaz de cumplir con su tarea de automatizar un proceso, sino que también debe ser capaz de integrase en la industria, compartir información e interactuar con otros robots y personas.

Según la norma ISO 8373:2021 [8], se entiende por robot a un “mecanismo accionado programado con un cierto grado de autonomía capaz de realizar locomoción, manipulación o posicionamiento.” A diferencia de la anterior versión de esta norma, este concepto de robot se ha generalizado para recoger la gran cantidad de mecanismos robóticos existentes, sin necesidad de que todos cumplan con las mismas características de construcción o que desenvuelvan una misma tarea en común. En la misma norma, se especifica el concepto de colaboración como la “operación realizada por robots diseñados para su propósito y personas que trabajan dentro del mismo espacio.”

Los robots colaborativos han supuesto una solución importante para su implementación en procesos industriales, siendo capaces de realizar sus tareas en el mismo espacio en la que trabajan los operadores sin la necesidad de instalar implementos de alto nivel de seguridad,

puesto a que se han diseñado para interactuar con las personas, aportando de esta manera al tercer pilar de la industria 4.0 descrito anteriormente.

Además de la colaboración con personas, otro concepto clave es la cooperación entre robots, del cual nace el concepto de Sistema Multi Robot, donde múltiples robots realizan actividades de manera coordinada como locomoción y manipulación para una tarea o misión que un solo robot no podría conseguir [9].

1.5.3 Estado del arte

El uso de los SMR hoy en día se ha extendido a diferentes áreas más allá de la investigación, es así como existen aplicaciones en soluciones agrícolas, de logística, mantenimiento, búsqueda y rescate, entre muchos otros escenarios en los que se busca aprovechar de las capacidades de los robots para mejorar la eficacia y eficiencia de los procedimientos mediante la automatización de estos. No obstante, existen diversos desafíos [5] que estos sistemas aún deben superar y por lo cual tienen un auge en temas de investigación, siendo objeto de estudio en diferentes investigaciones relacionadas con temas de planificación de tareas, arquitecturas de control, integración de nuevas tecnologías (realidad virtual, realidad aumentada y realidad mixta), interfaces e interacción entre robots y con operadores, configuración de flotas, entre otros temas relevantes al momento de configurar un SMR y que depende del alcance de cada solución.

Según Lima y Custódio [10], los retos más relevantes que son tema de investigación en los SMR se resumen en la incertidumbre en los valores obtenidos de los sensores y en el resultado como tal de las acciones que realizan los agentes, la complejidad añadida (en temas de razonamiento, planificación, asignación de tareas, programación, control y aprendizaje) debido a la necesidad de los robots de cooperar y coordinar acciones, la calidad de comunicación que tiende a empeorar al aumentar el número de miembros de la flota de trabajo,

y finalmente, la integración de diferentes tecnologías que manejan los subsistemas de cada robot.

Relacionado con el primer reto descrito por Lima y Custódio, existen diferentes investigaciones relacionadas con el diseño de un sistema de control óptimo para un SMR y el manejo de las incertidumbres, de esta manera Zhang et al. en su artículo *Decentralized Control of Multi-Robot System in Cooperative Object Transportation Using Deep Reinforcement Learning* [11], muestran sus resultados al implementar controladores descentralizados *deep Q-network* (DQN) en un SMR con el objetivo de mejorar la cooperación de los robots en tareas de transporte, además de probar que una arquitectura descentralizada resulta ser más eficiente que su contraparte, y que el sistema es lo suficientemente robusto para manejar incertidumbres de nivel medio y pequeños. Por otro lado, Shule et al. [12], manejaron las incertidumbres de localización implementando un sistema basado en Ultra-wideband (UWB), una tecnología reciente que se ha convertido en una solución robusta para problemas de localización en GNSS denied environments (ambientes donde la comunicación entre los satélites y los robots no es consistente), concluyendo que el uso de UWB puede convertirse en una tecnología estándar para el posicionamiento en SMR.

Para manejar a complejidad añadida, existen investigadores que estudian y proponen diferentes soluciones relacionadas con la planeación de caminos o trayectorias en SMR. Mutawe et al. [13] por ejemplo proponen un algoritmo de control para el seguimiento de trayectorias y coordinación de caminos en un SMR conformado por drones y robots terrestres, buscando evitar la colisión entre ellos mientras realizan sus trayectorias mediante la implementación de controladores PID en cada robot. Matoui et al. [14], por otro lado, introduce en el sistema un “supervisor” el cual se encarga de realizar los cálculos necesarios y controlar los robots, desarrollando así una arquitectura centralizada para la planeación de caminos en un SMR. Finalmente, Madridano et al. [15], en su artículo repasan diferentes métodos y algoritmos

para la planeación de trayectorias, enfocándose en su implementación alrededor de SMR y en soluciones ya estudiadas.

En el tema de comunicación, Klavins [16] en su reporte *Communication Complexity of Multi-robot Systems*, estudia la escalabilidad de los algoritmos utilizados en SMR, considerando el nivel de coordinación entre los agentes e introduciendo el concepto de complejidad en la comunicación de estos sistemas y concluyendo que no es sencillo determinar la complejidad mínima que presenta el sistema al realizar una tarea compleja. Otros estudios se encaminan en probar y evaluar otras soluciones de comunicación, como Chen et al. [17], quienes estudian la factibilidad de implementar una red inalámbrica para la comunicación de un SMR en una denominada “fábrica inteligente”, buscando incorporar flexibilidad, mejorar la productividad y aprovechar de manera eficiente la energía con la que cuenta la fábrica.

Capítulo 2

2.1 Metodología.

En el presente capítulo se describen las alternativas de solución para el problema planteado en el capítulo anterior y se establece una solución a desarrollar en función de ciertos criterios de diseño. Posterior, se describe de manera más detallada la solución, enfocándose en los componentes computacional y de control que requirió este sistema mecatrónico, detallando decisiones de diseño y componentes a utilizar.

2.2 Requerimientos de diseño.

Durante el desarrollo del proyecto y mediante la interacción con el cliente, se establecieron una serie de requerimientos para la solución desarrollada, condicionando elementos que deben estar presentes, tareas que se deben ejecutar y mejoras que se pueden realizar, todo según las necesidades del usuario, los cuales se resumen en la Tabla 2.1.

Tabla 2.1

Requerimientos de diseño del proyecto

Concepto	Determinaciones
Escalabilidad	El sistema multi-robot debe ser capaz de manejar 3 robots, para posteriormente ir aumentando el número de robots a medida que se adquieren o construyen más robos móviles.
Robustez	El sistema debe ser capaz de detectar las posiciones de los agentes.
Robustez / Experiencia de usuario	Los agentes deben ser capaces de evadir obstáculos y evitar colisionar con las personas que trabajen en el laboratorio.
Comunicación	Se debe integrar diferentes robots y sensores, obteniendo la información necesaria de cada uno de ellos e indicar tareas a realizar.
Escalabilidad / Robustez	El laboratorio cuenta con cámaras, pudiendo ser utilizadas en el sistema de ser necesario.
Comunicación / Experiencia de usuario	Los robots deben cumplir con ciertas tareas de transporte de objetos, para lo cual el framework debe ser capaz de elegir a un robot para la tarea según especificaciones como la cercanía.

Nota. Descripción de los requerimientos de diseño.

2.3 Solución

2.3.1 Alternativas de solución

A partir de los requerimientos de diseño identificados en la sección anterior y la revisión bibliográfica realizada en el estado del arte se definieron las siguientes alternativas de solución:

- **Alternativa 1:** Framework multi-robot de arquitectura centralizada para el control de los agentes, con un máster que recepte la ubicación estimada por los sensores propios del robot y asigne las tareas a realizar.
- **Alternativa 2:** Framework multi-robot de arquitectura descentralizada constituido por nodos traductores de información para la intercomunicación entre agentes y nodos de decisión para la asignación de tareas.
- **Alternativa 3:** Framework multi-robot de arquitectura centralizada que integre agentes y sensores externos con un módulo propio de localización de agentes y un máster de asignación de tareas.

Las principales diferencias entre las alternativas declaradas se resumen en la jerarquía del sistema, pudiendo ser centralizada o descentralizada, y el uso de los sensores propios de cada robot o de la integración de sensores externos.

2.3.2 Criterios de selección

Con el fin de determinar la alternativa de solución que más se ajuste a los requerimientos estipulados se consideraron los siguientes criterios de evaluación:

- **Viabilidad:** Se consideró la posibilidad de construcción o programación del framework.
- **Robustez:** Se evaluó la capacidad del sistema para tratar las incertidumbres generadas en la localización de los agentes y espacios de trabajo.
- **Escalabilidad:** Se evaluó la capacidad de integrar más agentes a futuro.

- **Modularidad:** Se consideró el encapsulamiento de los diferentes subsistemas y su integración en el framework.
- **Comunicación:** Se evaluó la calidad en la comunicación entre los agentes.
- **Capacidad de respuesta:** Se evaluó la velocidad de respuesta tras requerir una tarea.
- **Experiencia de usuario:** Se consideró la experiencia del usuario final al momento de ejecutar el framework, integrar nuevos agentes o sensores, y la seguridad de operación.

En la Tabla 2.2 se enumeran los criterios de evaluación ordenados en función de su relevancia en el diseño de la solución, además se asignaron los pesos respectivos.

Tabla 2.2

Criterios de evaluación

Criterio	Relevancia	Peso
Escalabilidad	1	6
Robustez	2	5.5
Experiencia de usuario	3	5.5
Capacidad de respuesta	4	4.5
Comunicación	5	4
Modularidad	6	3
Viabilidad	7	2

Nota. Criterios de evaluación ordenados por relevancia y peso.

2.3.3 Matriz de decisión

Posterior al establecimiento de los criterios de evaluación se procedió a realizar una matriz de decisión para comparar las alternativas de solución propuestas y así determinar la solución a desarrollar en el presente proyecto. Los resultados obtenidos se detallan en la siguiente Tabla 2.3.

Tabla 2.3

Matriz de decisión para la selección de alternativa

	Peso	Opciones	Alternativa 1	Alternativa 2	Alternativa 3
Criterio	6	**Escalabilidad**	3	2	3
	5.5	**Robustez**	1	2	3
	5.5	**Experiencia de usuario**	1	1	2
	4.5	**Capacidad de respuesta**	3	1	3
	4	**Comunicación**	3	2	2
	3	**Modularidad**	2	1	3
	2	**Viabilidad**	3	1	2
Resultados		**Puntaje sin peso**	16	10	18
		Puntaje con peso	66.5	46	80
		Prioridad	**2**	**3**	**1**

Nota. Matriz de decisión donde se evidencia la selección de la alternativa 3.

A partir de los resultados obtenidos, se definió como solución a la problemática planteada la alternativa 3, la cual proporciona un sistema escalable y lo suficientemente robusto para trabajar con cualquier número de robots/agentes que se desee, además de manejar de mejor manera las incertidumbres al incorporar sensores externos que proporcionan información del entorno de trabajo, lo que a su vez brinda mayor seguridad a los operadores y usuarios, además resulta ser más escalable al manejar un sistema centralizado, por lo que solo es necesario intercomunicar al nuevo agente o sensor con el sistema central.

La alternativa 1 carece de la robustez de la alternativa 3 al utilizar únicamente los sensores propios de los agentes, por lo que se convierte en la segunda prioridad, mientras que la alternativa 2 resulta más complicada de ejecutar y menos escalable, puesto que al manejar una arquitectura descentralizada no se establece una vía de comunicación eficiente dado a que todos los robos deben intercambiar información entre ellos mismos al mismo tiempo, aunque

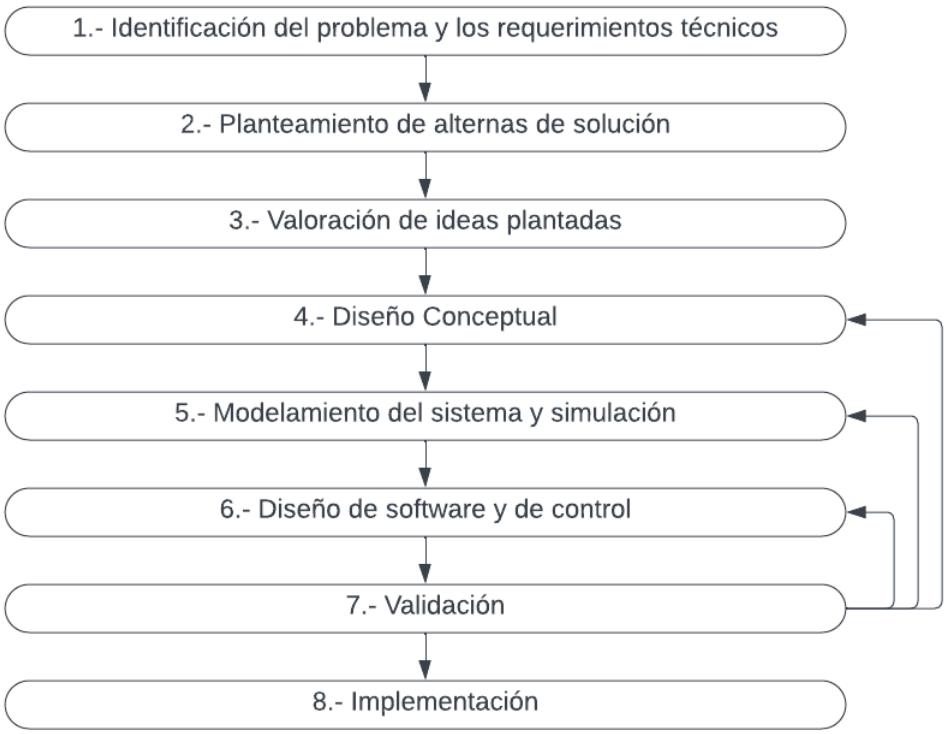
esto aumenta el uso de los sensores de cada robot y disminuye la incertidumbre de los datos. En general, cada uno de los sistemas tiene sus ventajas y desventajas, sin embargo, la alternativa 3 resulta más equilibrada y proporciona un sistema más robusto, aunque necesita de sensores extras.

2.4 Proceso de diseño

Se siguió un proceso de diseño iterativo, como se muestra en la Figura 2.1. La primera fase consiste en la selección de una solución. Primero, se identificó el problema y los requerimientos técnicos del cliente y del sistema. Posteriormente, se plantearon alternativas de solución al problema. Finalmente, se valoraron las soluciones planteadas considerando criterios de evaluación en una matriz de decisión para escoger la solución óptima. Luego, se siguió un proceso iterativo entre los pasos 4, 5, 6 y 7 de la Figura 2.1. Se realizó un diseño conceptual de la solución. Posteriormente, se modeló el sistema en el entorno de simulación de Gazebo y se realizó el diseño de software y de control. Al validar el diseño realizado en la simulación, se puede requerir volver a las etapas de diseño conceptual, modelamiento o diseño de software y control, debido a posibles errores o buscando optimizaciones, A la par, se realizaron ciertas pruebas en el entorno real de algunos módulos del framework.

Figura 2.1

Proceso de diseño



Nota. La figura muestra el proceso de diseño seguido durante el proyecto.

2.5 Diseño conceptual

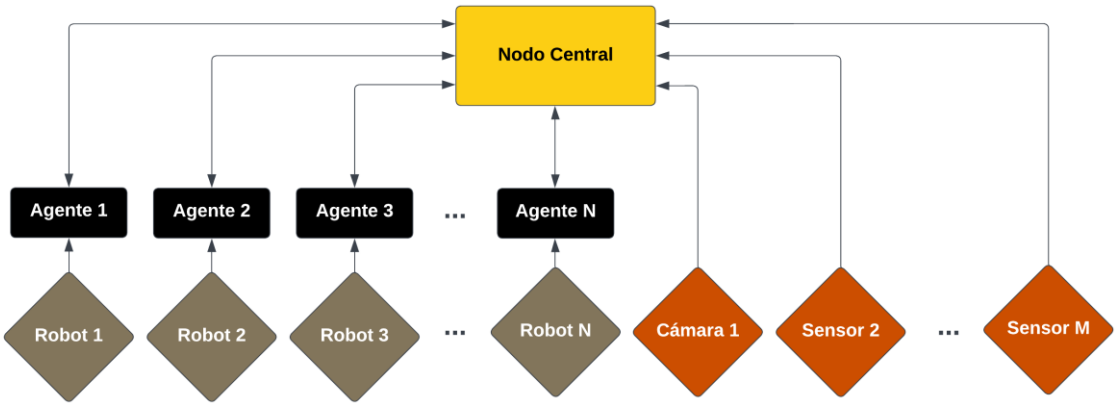
En el diseño conceptual del framework de SMR, el sistema se compone de un nodo central que se encarga de manejar y coordinar a los distintos agentes y sensores. El nodo central funciona como cerebro del sistema y es el responsable de tomar decisiones de alto nivel, realizar distribución de tareas y manejo de recursos. El nodo central recopila la información obtenida por los agentes y sensores y la procesa para tomar decisiones basándose en algoritmos de control. Utilizar un sistema centralizado permite un enfoque eficiente para la coordinación de varios robots.

Por otro lado, los agentes y sensores son entidades físicas involucradas con el sistema. Los agentes son los encargados de completar las tareas requeridas y recopilar información del entorno, pueden ser robots equipados con sensores para localización y percepción; mientras

que existen sensores externos como cámaras para realizar visión por computadora u otros dispositivos IoT relevantes para la aplicación escogida. La comunicación entre los agentes y el nodo central es bidireccional, mientras que las cámaras y sensores tienen una comunicación unidireccional; como se muestra en la Figura 2.2.

Figura 2.2

Agentes del SMR



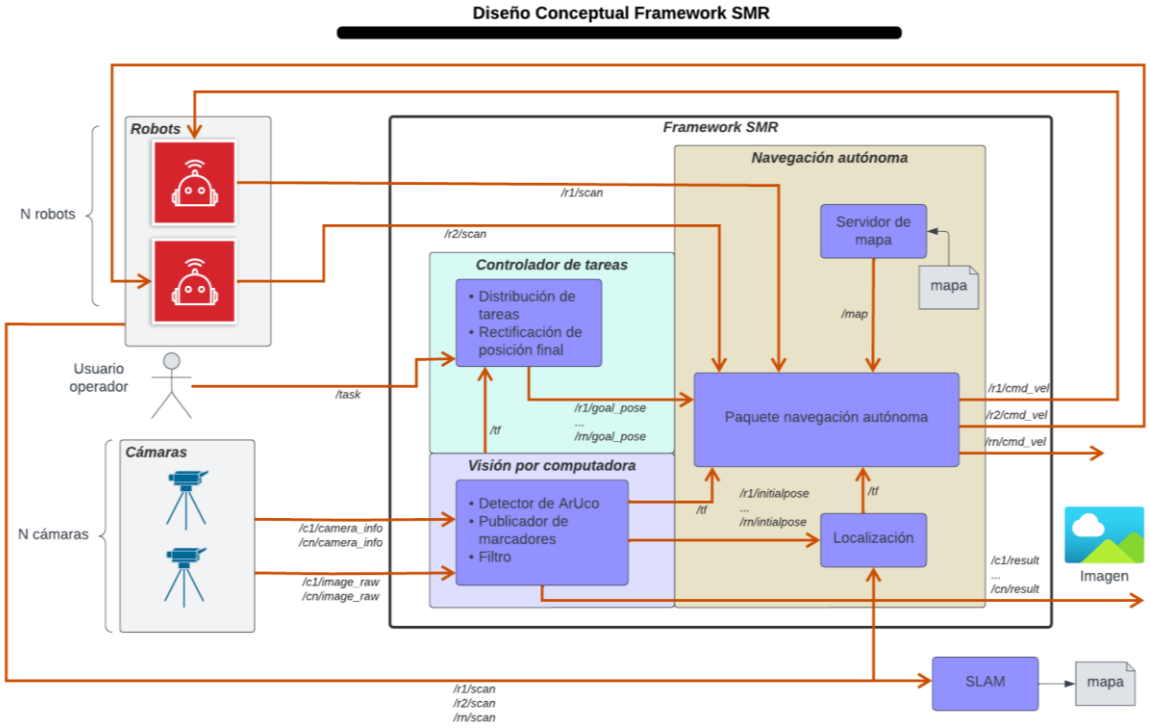
Nota. La figura muestra la arquitectura del framework solución.

El framework de SMR consiste en 3 principales módulos: navegación autónoma, distribución de tareas y visión por computadora. La Figura 2.3 muestra un diagrama conceptual del framework de SMR. Las entradas del sistema son los comandos del operador para realizar tareas y tópicos de un número indefinido de robots y de cámaras. Los tópicos de los robots que entran al framework son los referentes a la información del sensor LiDAR de cada robot. Previamente, se debe realizar el mapeo del ambiente de trabajo y guardarlo en un archivo. Posteriormente, se ejecuta un servidor de mapa para alojar la información del archivo del mapa en un tópico de ROS. El módulo de navegación recibe la información del mapa, del LiDAR de cada robot, las transformadas de localización de cada robot y el objetivo o pose final requerida. La salida del módulo de navegación son los comandos de velocidad que se enviarán a los controladores de los robots para poder seguir las trayectorias calculadas.

Por otro lado, los tópicos de las cámaras contienen datos acerca de la cámara y la imagen sin procesar obtenida. Con esta información se ejecutan los nodos para detectar marcadores ArUco y publicar sus transformadas. Además, se añadió un filtro para solo publicar la transformada obtenida de la cámara más cercana al ArUco detectado. Los marcadores ArUco se utilizan para detectar las poses de los robots y de las estaciones de trabajo del ambiente. Las salidas de este módulo son una imagen resultante que contiene la detección de marcadores ArUco y las posiciones iniciales para el módulo de localización probabilística. Finalmente, se desarrolló un módulo de distribución de tareas que recibe los comandos del operador y consigna la tarea al robot mejor preparado tomando en cuenta varios criterios, entre los que se prioriza la cercanía al destino.

Figura 2.3

Diseño Conceptual Framework SMR



Nota. La figura muestra la estructura general del framework.

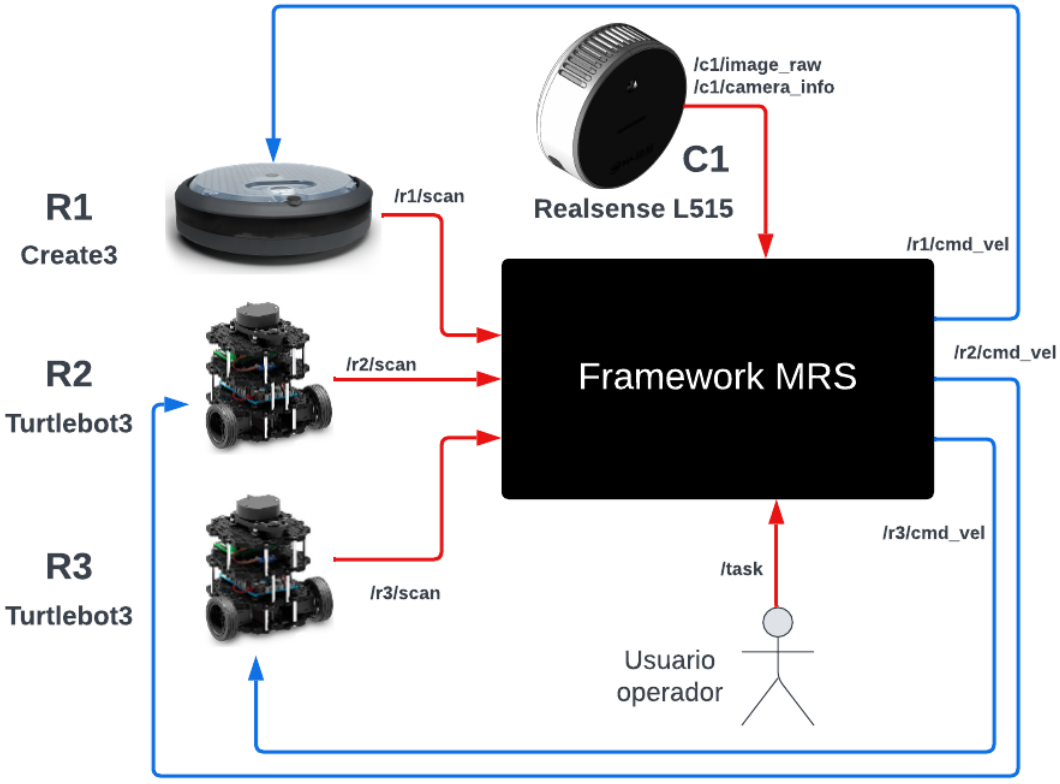
2.6 Diseño informático

2.6.1 Entorno de simulación

Durante las fases de diseño y validación concurrente del framework, se realizaron simulaciones en Gazebo. Gazebo es un entorno de simulación que permite emular escenarios reales, por lo que se pudo trabajar y mejorar cada módulo del framework antes de implementarlo completamente en la simulación del espacio de RAMEL. La simulación en Gazebo incluye 2 Turtlebot3 y un Create3, robots que dispone el cliente para su ambiente colaborativo, además de cámaras Intel Realsense L515, tal como se muestra en la Figura 2.4.

Figura 2.4

SMR RAMEL



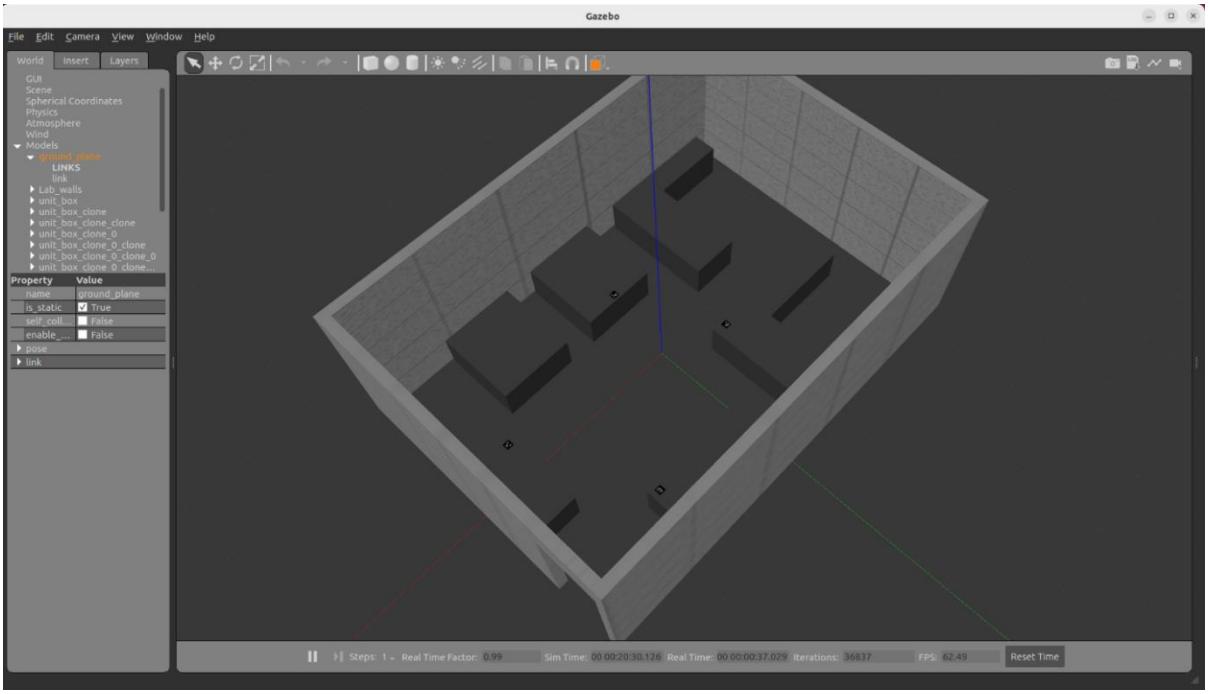
Nota. La figura muestra la estructura del framework a implementar en RAMEL.

Se creó un mundo simulado para representar de manera realista el laboratorio, incluyendo obstáculos y disposiciones espaciales representativas. De esta manera, se puede

evaluar el rendimiento del sistema en un contexto cercano al real. En la Figura 2.5, se observa la representación que se creó de RAMEL en el entorno de simulación de Gazebo.

Figura 2.5

Representación RAMEL en Gazebo



Nota. La figura muestra una representación del interior del laboratorio RAMEL para realizar pruebas.

2.6.2 Manejo de varios robots en ROS2 (namespaces)

Debido a que se administraron varios robos, se utilizaron namespaces en ROS2 para lograr una coordinación y comunicación eficientes. A cada robot se le asignó un namespace, el cual encapsula los tópicos, nodos y servicios de dicho robot evitando conflictos de nombres e información. Los namespaces que se utilizaron siguen la estructura r[n]. Es decir, el primer robot agregado al SMR tiene el namespace “r1” y así progresivamente.

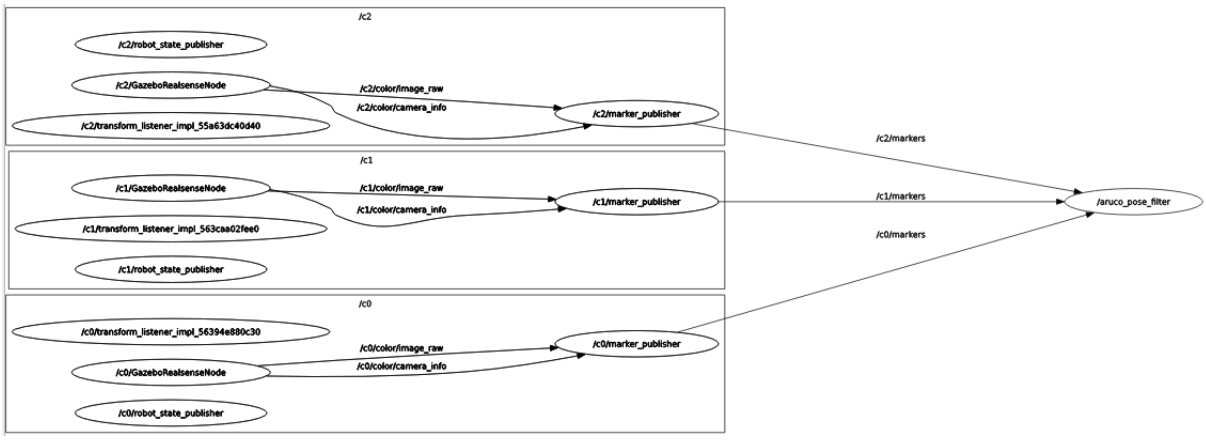
Gracias a los namespaces, los tópicos son identificados de manera única por su prefijo. Esto permitió la comunicación entre robots sin interferencias. Por ejemplo, el tópico llamado “/r1/scan” corresponde a la información captada por el LiDAR del robot 1, mientras que el tópico “/r2/scan” es referente a los datos del LiDAR del robot 2. Además, con el uso de

namespaces se obtuvo una mayor modularidad y reusabilidad, debido a que los nodos desarrollados para un robot pueden ser fácilmente adaptados a los otros solo modificando el namespace. De la misma manera, se manejaron namespace para las cámaras añadidas en la simulación con la referencia c[n].

En la Figura 2.6 se muestran los nodos correspondientes al a 3 cámaras, encapsulados dentro de los namespaces correspondientes c0, c1 y c2. Además, los tópicos (representados por flechas) también llevan el identificador.

Figura 2.6

Encapsulación cámaras

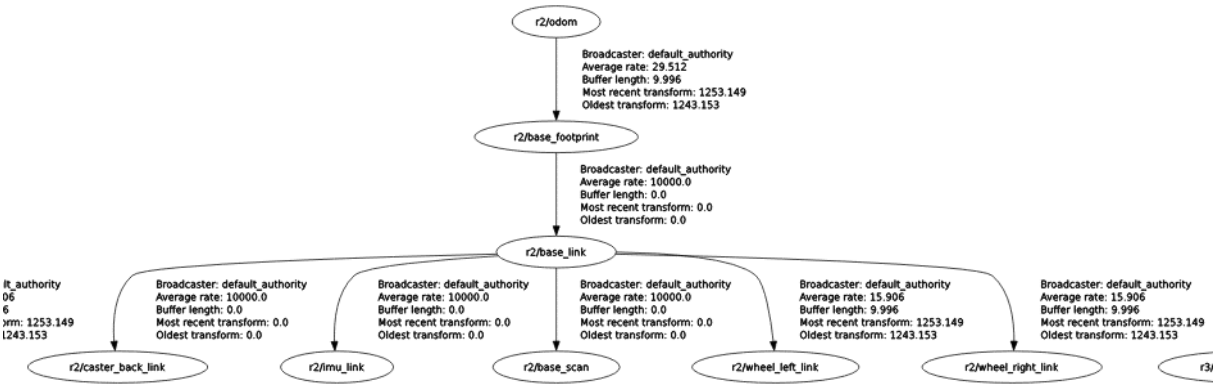


Nota. Figura obtenida a partir de la simulación en ROS – Gazebo.

No obstante, los marcos de coordenadas o transformadas TF no se ven afectados por el namespace del robot. Por esta razón, se decidió utilizar prefijos de transformadas con la misma estructura que los namespaces. De esta forma, no se tendrán conflictos con las referencias de cada robot. Por ejemplo, el árbol de transformadas para el robot 2 queda como se muestra en la Figura 2.7.

Figura 2.7

Árbol de transformadas del robot 2



Nota. Figura obtenida a partir de la simulación en ROS – Gazebo.

2.6.3 Mapa del ambiente colaborativo

Para que los robots puedan navegar de forma autónoma, se requirió de un mapa del entorno en el cual se encuentran. Por lo tanto, se utilizaron sensores LiDAR para poder recolectar información del entorno y construir un mapa con técnicas de SLAM. SLAM es el proceso en el cual un robot móvil puede construir un mapa del entorno y, de forma simultánea, usar el mapa para obtener su ubicación [18]. El proceso de obtención de mapa se puede realizar con cualquier paquete de ROS2, o incluso con algún método independiente a ROS2. Se debe tener como resultado final para utilizar el framework un archivo de parámetros del mapa en formato ".yaml" y una imagen del mapa que puede ser en formatos ".png", ".jpg", ". pgm", entre otros.

2.6.4 Localización con visión por computador

Uno de los desafíos presentes al momento de implementar un SMR es la incertidumbre de los datos la cual, como se describió en el capítulo anterior, se refiere a la falta de precisión y poca confiabilidad de la información que puede proveer un robot, como su ubicación exacta. Existen múltiples soluciones desde el uso de sistemas GPS hasta nuevas técnicas como Visual-

Inertial Odometry (VIO) y Simultaneous Localization and Mapping (SLAM) [19], no obstante, para ambientes cerrados existe la posibilidad de utilizar un sistema de visión por computador, haciendo uso de cámaras para la adquisición de imágenes. De este modo, el diseño del framework multi-robot del presente proyecto muestra un componente de visión por computadora, buscando proveer al sistema de una fuente extra de información del entorno.

La visión por computador, o también conocida como visión artificial, es una rama de la inteligencia artificial que busca dotar a un computador la capacidad de recibir y procesar información de su entorno en forma de imágenes, tal cual como un ser humano. El proceso, de manera resumida, consta de las siguientes etapas: Adquisición de imágenes, Procesamiento, Segmentación, Representación y descripción, y Reconocimiento e interpretación [20]. Esta tecnología ha mostrado un gran avance en los últimos años a la par que el desarrollo de las redes neuronales, siendo posible reconocer diferentes elementos presentes en una imagen según características como los colores, texturas, líneas y otros rasgos que el sistema detecta para establecer diferencias y reconocer la presencia o no de un objetivo.

Para el presente proyecto, el uso de la visión artificial radica en la localización de los robots y puestos de trabajos, por tal motivo se requiere de un sistema rápido que pueda proveer de información de manera online al framework multi-robot, que sea capaz de identificar posición y orientación de los robots, que pueda proveer de datos de manera precisa y con un mínimo margen de error, y que sea escalable a múltiples robots diferentes entre sí.

Con base en estos requerimientos, un sistema de visión por computador común que sea entrenado para reconocer robots en una imagen, que deduzca la posición a partir del tamaño del robot en la imagen, y que reconozca espacios de trabajo en un ambiente cerrado, aunque es posible, requiere de mucha inversión en cuanto a datos preliminares de entrenamiento y aun así no proveería de información lo suficientemente precisa y exacta para que el sistema sea

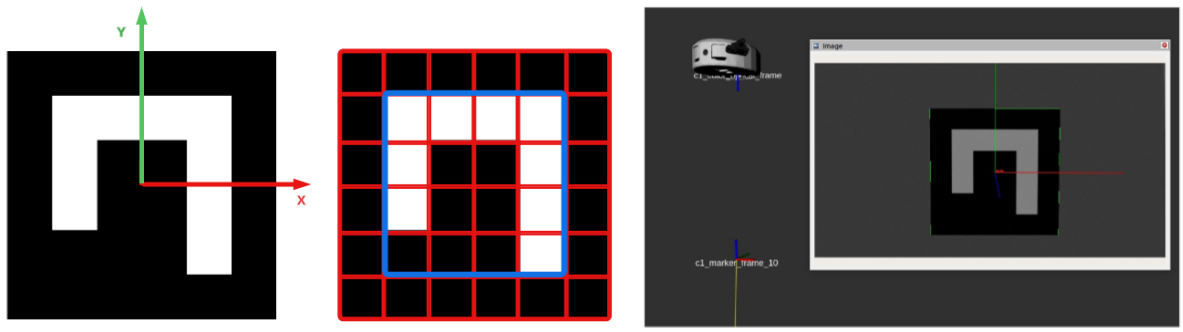
robusto. Por este motivo se evaluó la opción de implementar un sistema de visión artificial basado en marcadores ArUco.

2.6.5 Marcadores ArUco

ArUco es una librería de código abierto desarrollada por Rafael Muñoz y Sergio Garrido escrita en C++ y orientada a la detección de marcadores o indicadores fiduciales en imágenes, además de proveer la capacidad de estimar la posición de estos con respecto a la cámara, teniendo como fin su aplicación en realidad aumentada y en localización de robots [21]. Esta librería permite generar indicadores e implementar un sistema de detección y posicionamiento de estos. Para su implementación dentro del framework multirobot se utilizará el código proporcionado por la librería de visión por computador OpenCV [22], la cual hace uso del modelo original de ArUco además de otros componentes de visión artificial para la detección de los marcadores.

Figura 2.8

Marcadores ArUco



Nota. (a) Estructura de un marcador ArUco. (b) Celdas de un marcador de tamaño 4x4 con borde. (c) Detección del marcador ArUco ID 10 del diccionario 4X4.

Los indicadores ArUco consisten básicamente en códigos 2D con un patrón binario único para cada indicador, lo que permite su identificación, tal como se visualiza en la Figura 2.8. Los marcadores ArUco se agrupan en diccionarios según la base utilizada para formar los patrones binarios, según el tamaño y el número de indicadores que contiene, de esta manera se

puede personalizar el diccionario según la aplicación, tomando en consideración que resulta más fácil detectar códigos de menor tamaño y diccionarios que contengan un conjunto pequeño indicadores.

El proyecto, al requerir de robustez y de escalabilidad, hace uso de un diccionario predeterminado de la librería: DICT_4X4_50, la cual consta de códigos binarios 2D de 4x4 bits con el fin de utilizar indicadores de fácil detección, con un total de 50 códigos ArUco que pueden ser utilizados, desde el indicador 0 hasta el 49, siendo de fácil acceso y manipulación por parte del usuario final, quien cuenta con diversas herramientas tanto internas del código como externas en la web para generar indicadores según sus necesidades, además de poder cambiar el diccionario cuando se requiera dotando de flexibilidad al sistema. Todo este módulo de visión artificial estará atado a cada cámara que se utilice en el SMR, y se configurará según las ecuaciones descritas en el Apéndice A.

2.6.6 Navegación autónoma

La navegación autónoma permite a los robots moverse de forma independiente para llegar a destinos definidos mientras esquivan obstáculos. Un módulo de navegación autónoma se compone de planeadores locales, globales y localización. Los planeadores globales generan el camino óptimo basándose en el mapa global desde la posición actual del robot hasta la meta especificada, utilizando el algoritmo A*. Por otro lado, el planeador local toma en consideración la cinemática del robot y la información del entorno captada por el sensor. Es decir, modifica la trayectoria del robot en tiempo real para esquivar obstáculos nuevos [23].

Se utilizó el paquete Nav2 para poder implementar la navegación autónoma. Por cada robot del sistema, se deben ejecutar los nodos respectivos del paquete con el identificador del robot. Además, se editaron los parámetros de configuración según cada robot; especificando los marcos de coordenadas (tf) y tópicos correspondientes del robot. Cómo se trata de un framework, se programó de tal forma que pueda haber n robots. Es decir, el usuario solo tendrá

que enviar como parámetro el número de robots móviles del ambiente colaborativo y se ejecutarán n módulos de navegación.

Las entradas al módulo de navegación son un archivo XML de BehaviorTree, las transformadas TF, el mapa y la información de los LiDAR. El nodo “BT Navigation Server”, que es el componente de más alto nivel, aloja el BehaviorTree para poder implementar los comportamientos de navegación deseados. El nodo de servidor de BT se comunica con el resto de los nodos modulares mediante acciones de ROS. Entre los nodos modulares hay un servidor para crear rutas más suaves, planeadores globales y locales, servidor de mapa, servidores de recuperación y la posibilidad de añadir más servidores propios. La salida del módulo de navegación son comandos de velocidad para los motores de los robots móviles [24].

2.7 Diseño de control

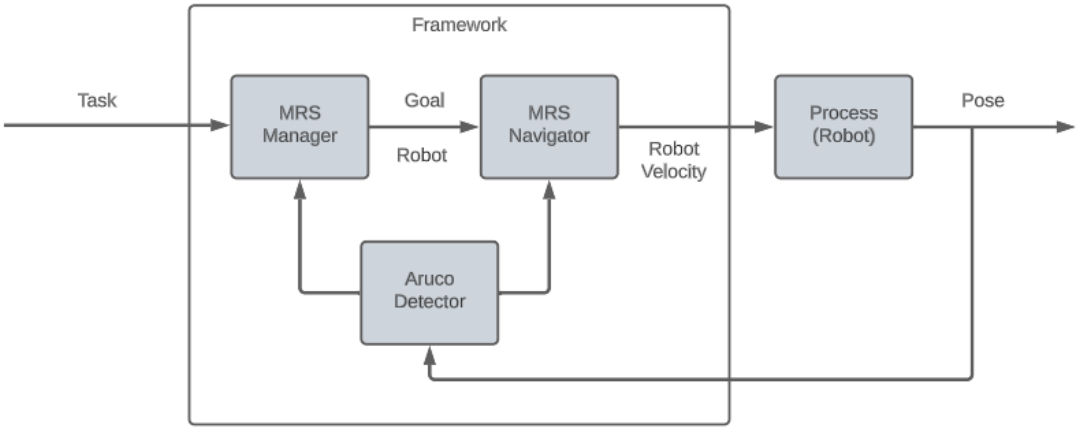
2.7.1 Algoritmo de control para navegación autónoma

El algoritmo de control utilizado para la navegación autónoma es un árbol de comportamientos. Un árbol de comportamientos es una arquitectura de control utilizada para definir el proceso de toma de decisiones de un agente de una manera modular. Se utilizó un árbol de comportamiento de navegación a objetivo con replanificación y recuperación, del paquete de Nav2, descritos en el Apéndice B.

De esta manera, el sistema de control resulta como se muestra en la Figura 2.9. En el cual el árbol de comportamiento es el algoritmo de control presente en el bloque MRS Navigator, que recibe la posición objetivo a partir del controlador de tareas y los comandos del operador. Además, gracias a la visión por computadora se recibe por retroalimentación las transformadas o posiciones actuales de los robots. La variable controlada por el algoritmo de control es el comando de velocidad hacia los controladores del motor de los robots, lo cual afecta el proceso o planta del sistema.

Figura 2.9

Sistema de control



2.7.2 Asignación de tareas

Se implementó un algoritmo de asignación de tareas multi-criterio basado en el estado de cada robot. El algoritmo considera la proximidad a la tarea y la disponibilidad del robot siguiendo la lógica presentada en la Figura 2.10. Además, se desarrolló el algoritmo de forma que sea adaptable y dinámico. Es decir, si el estado de un robot cambia u ocurren variaciones en el entorno, el algoritmo reevaluará la asignación de tareas.

Figura 2.10

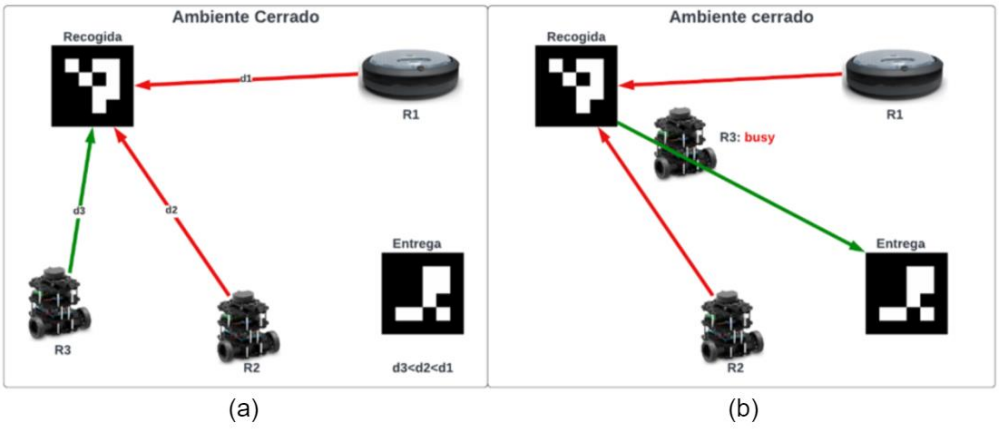
Pseudocódigo de asignación de tareas

```
Funcion robot_rn <- Buscar_robot_tarea ( numero_de_robot )
Fin Funcion

Funcion exito <- Asignar_tarea ( robot )
Fin Funcion

Algoritmo AsignarTareaCercano
    Leer lista_de_robots
    Leer lista_de_posiciones
    robot_tarea <- 0
    Para robot_i <- 1 Hasta numero_robots Con Paso 1 Hacer
        distancia_i <- distancia_destino-posición
        distancia_minima <- +inf
        Si (distancia_i < distancia_minima Y no robot_i_ocupado) Entonces
            distancia_minima <- distancia_i
            robot_tarea <- robot_i
        SiNo
            //Nada
        Fin Si
    Fin Para
    Si robot_tarea = 0 Entonces
        Imprimir "No hay robots disponibles"
    SiNo
        robot <- Buscar_robot_tarea(robot_tarea)
        asignado <- Asignar_tarea(robot)
        Imprimir "Tarea asignada a {robot}"
    FinSi
FinAlgoritmo
```

En la Figura 2.11 se muestra un ejemplo de asignación de tarea. Se envía una posición de recogida y de entrega, solicitando que un robot realice esta tarea. Dado a que el robot 3 es el más cercano a la posición de recogida (vector verde) y no se encuentra ocupado, la tarea es realizada por este.

Figura 2.11

Ejemplo de asignación de tarea



Nota. (a) Búsqueda del robot más cercano para asignación de tarea. (b) Ejecución de tarea.

Capítulo 3

3.1 Resultados y Análisis

En esta sección se muestran los resultados obtenidos del framework SMR inicialmente de forma individual por cada uno de los módulos del framework en un ambiente simulado del espacio de trabajo del laboratorio RAMEL. Posteriormente, se muestran SMR aplicados en diferentes ambientes simulados. Finalmente, se culmina la sección con un análisis de costos.

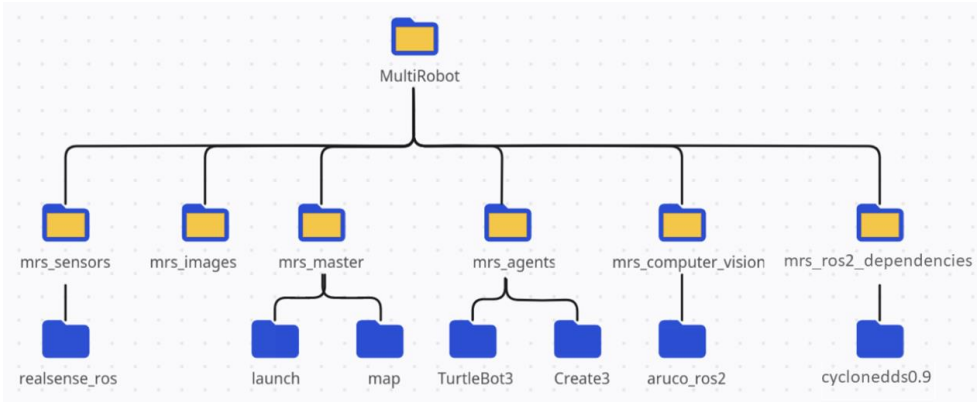
3.2 Arquitectura y comunicación.

3.2.1 Estructura del repositorio

A partir de los paquetes necesarios para la construcción del framework MultiRobot, se organizó un repositorio en GitHub de manera que se pueda diferenciar cada uno de los módulos y componentes, como se observa en la Figura 3.1. Así, se tienen las carpetas mrs_sensors y mrs_agents que contienen los paquetes de los sensores y robots, para este caso el paquete del modelo de la cámara Realsense y los paquetes de los robots TurtleBot3 y Create3. Por otro lado, la carpeta mrs_computer_vision contiene los paquetes para el reconocimiento de marcadores ArUco, mientras que la carpeta mrs_master contiene el manager de tarea, módulos de navegación y lanzadores de la aplicación, además del archivo del mapa del ambiente. Otras carpetas como mrs_images guardan imágenes del proyecto y mrs_ros2_dependencies contienen dependencias que fueron modificadas para SMR.

Figura 3.1

Estructura del repositorio del proyecto

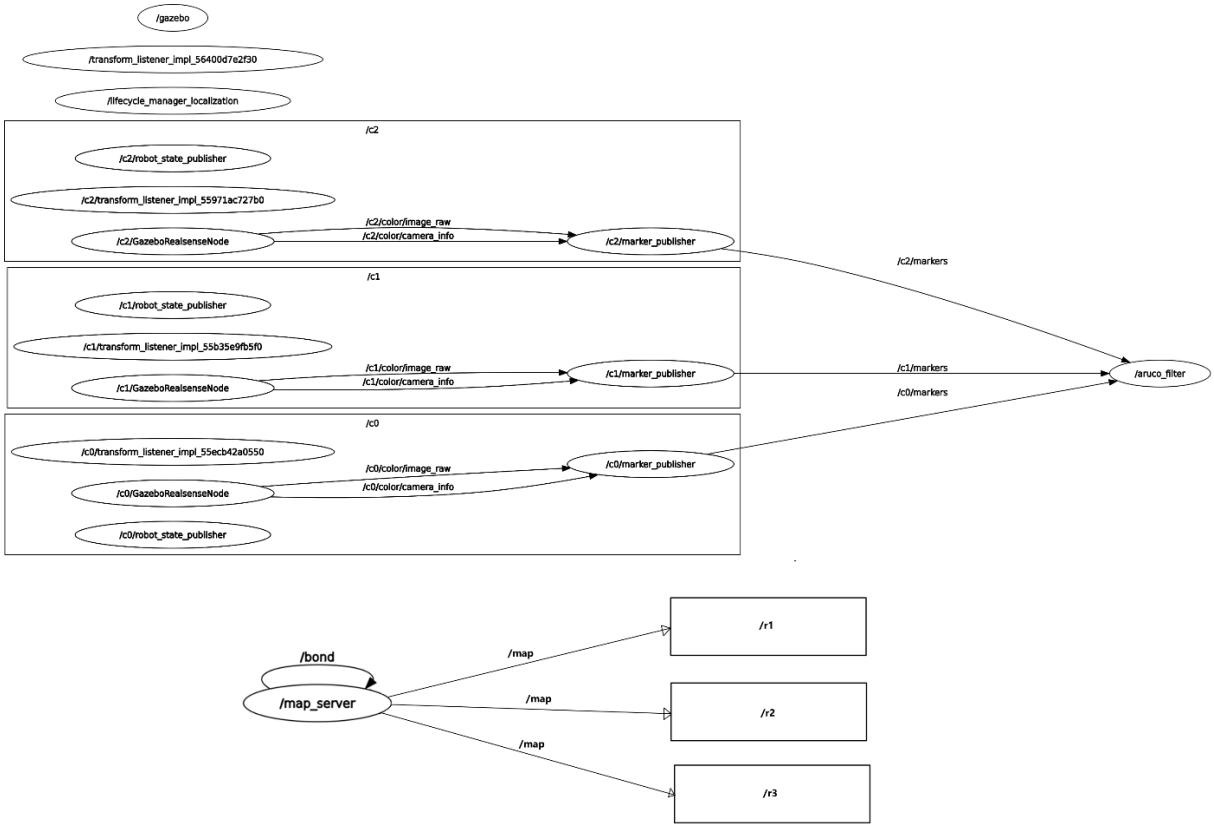


3.2.2 Estructura de comunicación de nodos

Se utilizó la herramienta de visualización de ROS2 ‘rqt’ para obtener una representación visual y concisa de la topología de nodos y la comunicación entre componentes en ROS2. El resultado obtenido se encuentra resumido en la Figura 3.2. En referencia a las cámaras, estas proporcionan los tópicos "cn/image_raw" y "cn/image_info". Estos tópicos son recibidos por el nodo "marker publisher", el cual tiene la función de procesar la información contenida en ellos y detectar marcadores ArUco. Posteriormente, el nodo transmite las detecciones en el tópico denominado "/cn/markers". Concluido este proceso, los tópicos generados por cada cámara convergen en el nodo "aruco filter", en este nodo se lleva a cabo la selección de la cámara más cercana para determinar la posición y orientación precisas del marcador ArUco en cuestión.

Figura 3.2

Gráfico de Nodos: Cámaras y Map Server

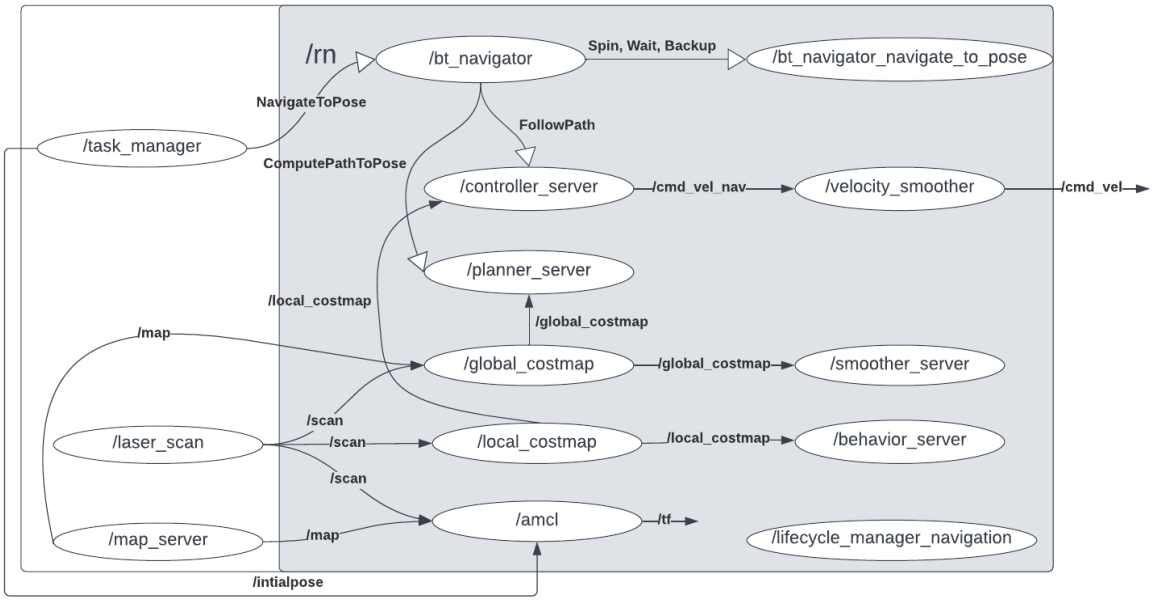


Una vez que se ha realizado esta selección, se publica la información correspondiente sobre la posición y orientación en el sistema. Por otro lado, los robots reciben la información del servidor de mapa a través del tópico “map”. Dentro de la encapsulación de cada robot, se dan los procesos de localización, planificación de rutas, navegador, servidores de comportamiento y control, entre otros. Dicha topología se ha desglosado de la figura con el propósito de permitir una visualización más comprensible y se abordará en una figura subsiguiente.

En la Figura 3.3 se muestra el gráfico de nodos para cada robot del SMR. El nodo de mapa de costo global recibe los tópicos de mapa y del LiDAR. Por otro lado, el nodo de mapa de costo local solo recibe la información del LiDAR. El nodo de localización amcl utiliza los tópicos “/scan” y “/map”, pero también reciben la posición inicial del robot del nodo administrador de tareas. Este nodo proporciona la transformada entre el marco del mapa y el marco de odometría del robot. El nodo administrador de tareas comanda acciones de tipo NavigateToPose al nodo “/bt_navigator”, dicho nodo es el máster de la navegación autónoma. Se comanda la acción “ComputePathToPose” al “/planner_server” para planificar rutas, la acción “FollowPath” al “/controller_server” para seguir la ruta planificada evadiendo obstáculos y acciones de recuperación tales como “Spin”, “Wait” y “Backup”. El nodo “lifecycle_manager_navigation” permite la comunicación entre todos los nodos mediante el tópico “bond”, todos los nodos que se encuentran en la zona gris de la figura están conectados entre sí por dicho tópico.

Figura 3.3

Gráfico de nodos simplificado para cada robot: Navegación



3.3 FrameWork MultiRobot

3.3.1 Interfaz Gráfica de Usuario

Se implementó una interfaz gráfica de usuario para facilitar el uso del framework para el usuario, proporcionando una experiencia intuitiva e interactiva así no tenga un conocimiento profundo en ROS2. En la Figura 3.4 se muestra el resultado final obtenido. El primer ítem marcado en la figura corresponde al botón de simular, que abre la simulación configurada. La simulación por defecto es la del ambiente de RAMEL con 2 Turtlebot3 y un robot create3. En el segundo ítem se le permite al usuario ingresar el número de robots y número de cámaras ya sea para simulación o para un sistema real. El ítem 3 es el botón para ejecutar el framework para el número de robots y cámaras definidos. El ítem 4 es el botón para inicializar el sistema, configurando las posiciones iniciales de los robots mediante visión por computadora. El ítem 5 se trata de 2 pestañas para designación de tareas. El usuario puede elegir entre las misiones de movilización de objetos o posición objetiva. Posteriormente, configurar los parámetros de

cada misión y enviar la tarea con los botones inferiores. En el ítem 6 se muestra una representación visual en tiempo real de la detección de marcadores aruco y con el ítem 7 se puede seleccionar que cámara se quiere vigilar.

Figura 3.4

Interfaz Gráfica de Usuario

3.3.2 Preparación de ambiente simulado

Se agregaron al ambiente simulado de RAMEL todos los componentes necesarios para evaluar el framework SMR. En primer lugar, los 3 robots fueron agregados utilizando paquetes de software existentes. Además, se agregaron cámaras y marcadores aruco. La Figura 3.5 muestra el resultado final del ambiente simulado.

Figura 3.5

Ambiente simulado RAMEL en Gazebo

Como se mencionó anteriormente, obtener el mapa del entorno es un requisito previo para utilizar el framework. Por lo tanto, se realizó el mapeo del ambiente simulado utilizando un turtlebot3, dicho proceso se muestra en la Figura 3.6. El mapa obtenido corresponde a la Figura 3.7.

Figura 3.6

SLAM del ambiente simulado

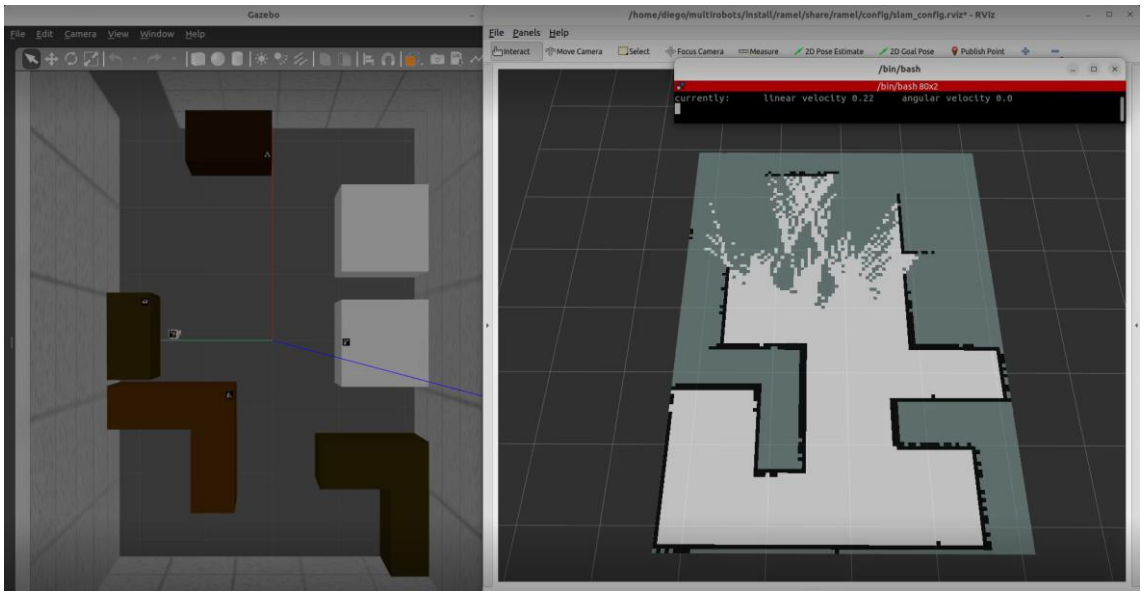
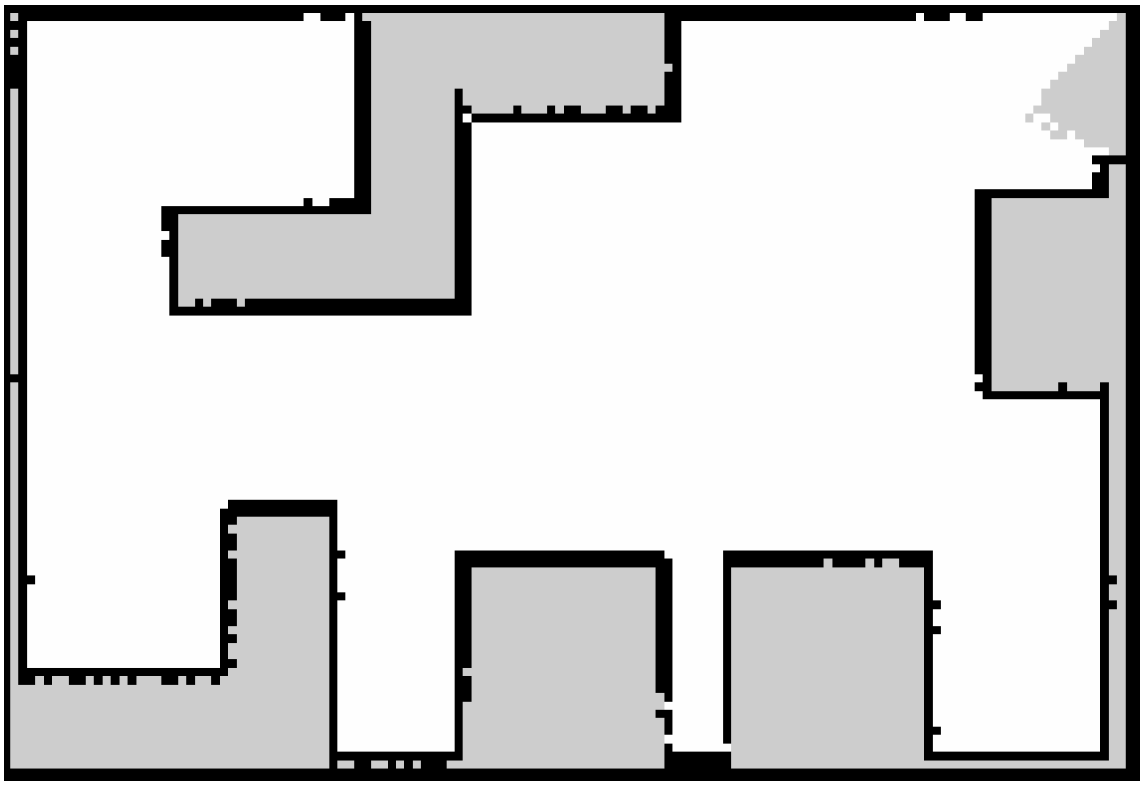


Figura 3.7

Mapeo realizado con SLAM



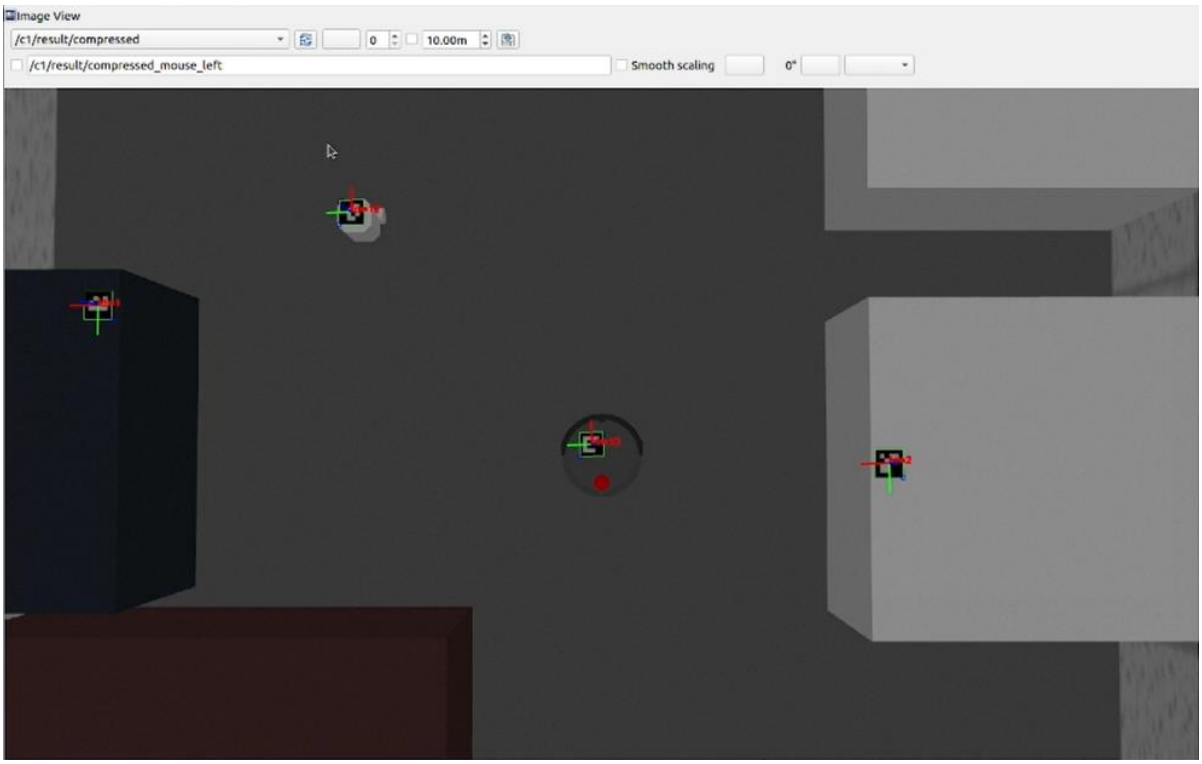
Nota. Mapa del ambiente obtenido con el paquete slam_toolbox.

3.3.3 Módulo de localización

Se agregaron al ambiente simulado marcadores arucos con el fin de reconocer la posición en tiempo real de los robots y los espacios de trabajo. Durante el arranque del Framework MultiRobot, este asocia a cada cámara del ambiente un nodo de reconocimiento de aruco, el cual procesa la imagen obtenida de la cámara para obtener la posición y orientación de los marcadores aruco que puede reconocer, estos marcadores deben pertenecer al diccionario 4x4 y se puede configurar como parámetros el tamaño real de los marcadores y la referencia global sobre la cual se calculará la distancia y orientación. En la Figura 3.8 se puede visualizar el resultado de la detección de los marcadores arucos realizado por la cámara c1, donde se incluyen 2 robots y 2 espacios de trabajo.

Figura 3.8

Detección y reconocimiento de marcadores ArUco en Gazebo

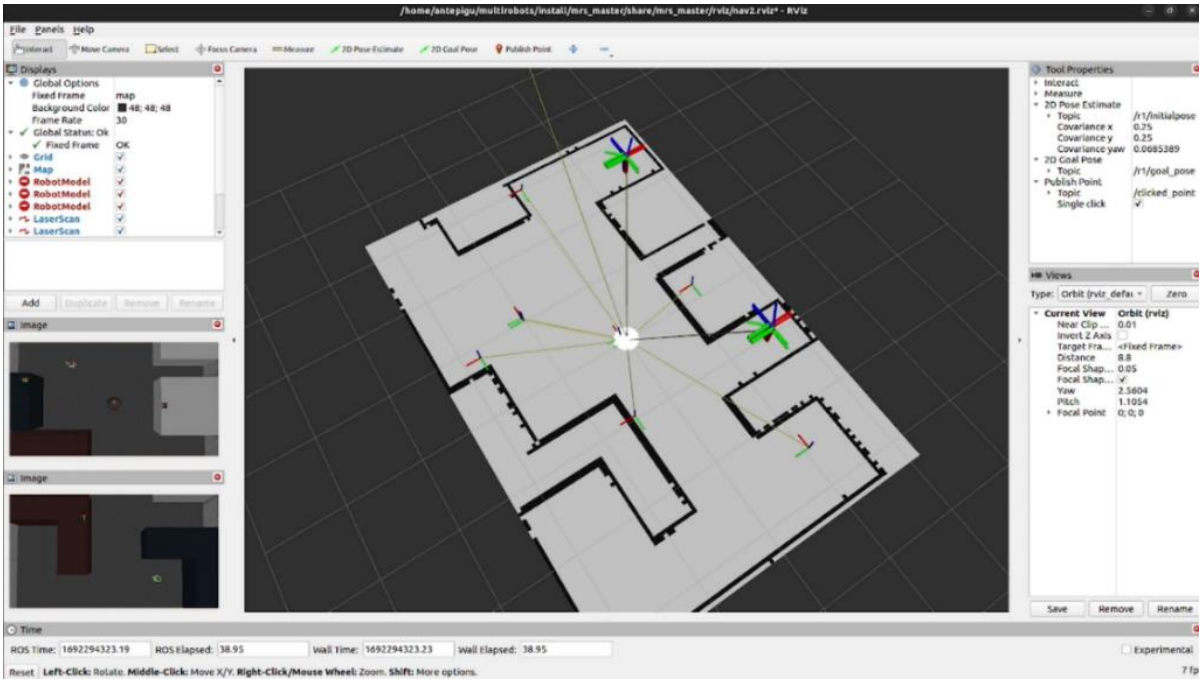


Una de las ventajas de utilizar marcadores ArUco es la generación de sistemas de coordenadas locales de cada marcador, de manera que al especificar un marco de coordenadas global de referencia en los parámetros se obtuvo la transformación de estos marcos de

referencia locales con relación a un origen predeterminado, en este caso el centro del mapa, visualizándose en la herramienta de RVIZ como se muestra en la Figura 3.9.

Figura 3.9

Visualización de los marcos de coordenadas en Rviz



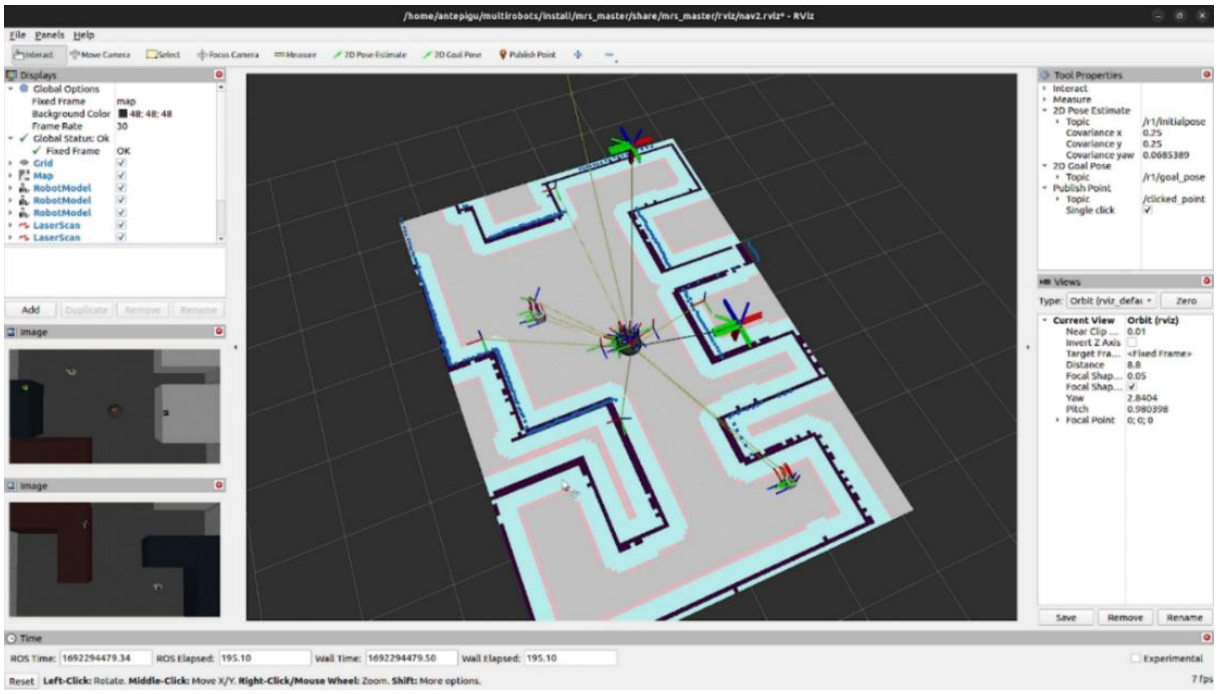
En la Figura 3.9 se puede identificar cómo el aruco de uno de los robots es identificado por dos cámaras al mismo tiempo mostrando dos marcos de coordenadas aproximadamente en la misma posición, para manejar estos casos se programó un nodo filtro de arucos, el cual permite obtener la información de cada nodo de reconocimiento de arucos perteneciente a cada cámara, reunir dicha información en una sola estructura o mensaje y procesar la información repetida para así simplificar la información necesaria y poder utilizarla para algunas funcionalidades del framework. En la Figura 3.2 se visualiza el gráfico de nodos resultante donde se evidencia la conexión de cada cámara con el filtro de arucos.

Una de las funcionalidades más importantes del framework es la retroalimentación de las posiciones y orientaciones de los robots, lo cual permite actualizar dicha información y corregir problemas de localización debido a factores externos que interfieran en el robot, esto se lo realiza a partir de la información del filtro de aruco, se seleccionan los arucos de los robots

y se procede a publicar el initial_pose para cada robot del ambiente. En la Figura 3.10 se puede apreciar la aparición de los robots en los lugares donde se habían reconocido la existencia de sus correspondientes marcadores arucos.

Figura 3.10

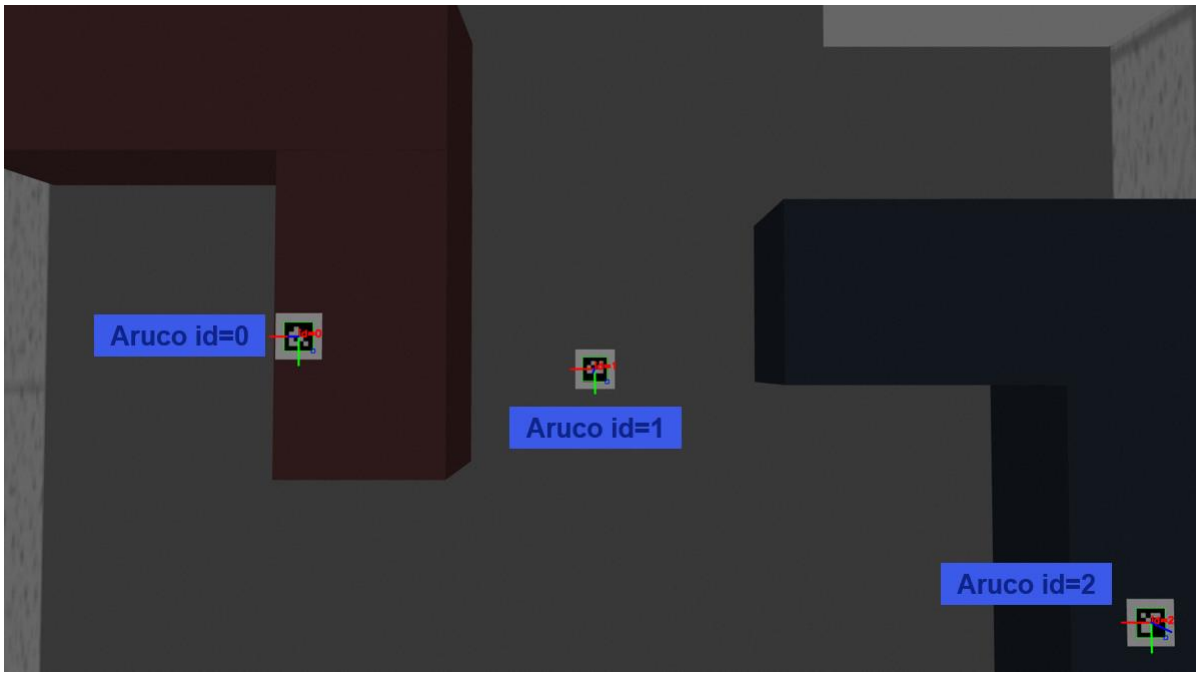
Inicialización de posición de los robots



El módulo de localización por visión artificial mediante arucos cuenta con un error dependiente de la cámara, su correcta calibración, la distancia a la que se encuentre el marcador y factores externos como la iluminación. A continuación, se calculó el error absoluto de las posiciones de los arucos en diferentes casos, visualizados por la cámara c0 como se muestra en la Figura 3.11.

Figura 3.11

Reconocimiento de marcadores ArUco por la cámara c0 en Gazebo



Para cada uno de los casos, como se pueden observar en la Figura 3.12, los modelos 3D de cada marcador en el espacio de simulación de Gazebo tienen su sistema de referencia desplazado a uno de los lados en el eje x sentido negativo, mientras que los marcadores identificados mediante el módulo de localización cuentan con sus sistemas de referencia ubicados en el centro del marcador, por tal motivo, para comparar los resultados, para las posiciones mostradas por Gazebo se sumará a la ubicación en el eje x el valor de 0.080 m correspondiente al desplazamiento antes señalado (0.05 m debido a la mitad del tamaño del marcador y 0.03 m del borde blanco). Las fórmulas de error a utilizar para cada uno de los ejes de coordenadas corresponden a las ecuaciones 3.1, 3.2, 3.3; mientras que el error de localización se obtendrá mediante la ecuación 3.4, considerando la posición de los marcadores en el plano XY. Por otro lado, el error de orientación se obtendrá mediante la transformación del quaternion resultante a ángulos de Euler mediante la herramienta https://www.andre-gaschler.com/rotationconverter/, siendo la fórmula del error de orientación la ecuación 3.5

correspondiente al valor absoluto de la diferencia entre el valor real (giro de 90° en el eje z) y el valor de rotación medido.

$$X_{error} = ||(pose\ x_{Gazebo} + 0.08) - pose\ x_{Rviz}|| \quad (3.1)$$

$$Y_{error} = ||pose\ y_{Gazebo} - pose\ y_{Rviz}|| \quad (3.2)$$

$$Z_{error} = ||pose\ z_{Gazebo} - pose\ z_{Rviz}|| \quad (3.3)$$

$$Error_{pose} = \sqrt{(X_{error})^2 + (Y_{error})^2} \quad (3.4)$$

$$Error_{rot} = |90° - Z_{angle}| \quad (3.5)$$

A continuación, en la Tabla 3.1 se resumen los resultados obtenidos en la detección de 3 marcadores ArUco realizado por la cámara c0.

Tabla 3.1

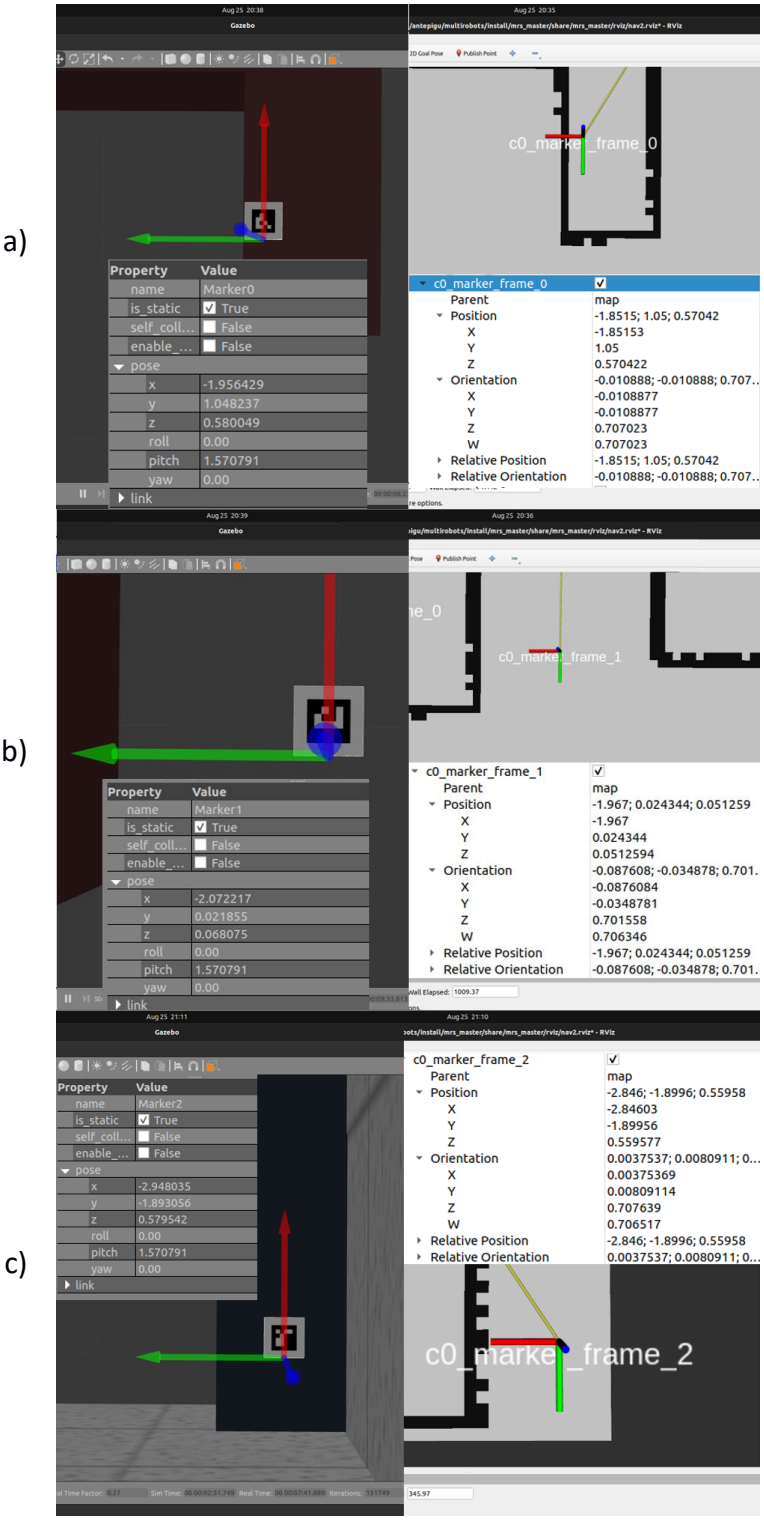
Resultados de error de posición en el reconocimiento de marcadores ArUco.

Marcador	Error de posición [m]	Error de rotación [°]
Id 0	0.0250	0.00
Id 1	0.0253	0.77
Id 2	0.0229	0.09
Promedio	0.0244	0.29

Nota. Los resultados se obtuvieron a partir de los datos de la Figura 3.12.

Figura 3.12

Localización real y detectada de los marcadores ArUco



Nota. Representación Errores: (X_{error}, Y_{error}, Z_{error}) (a) Resultados ArUco id 0: Errores: 0.0249 m, 0.0018 m, 0.0096 m. Z_{angle} = 90° (b) Resultados ArUco id 1: Errores: 0.0252 m, 0.0024 m, 0.0168 m. Z_{angle} = 89.23° (c) Resultados ArUco id 2: Errores: 0.0220 m, 0.0065 m, 0.0199 m. Z_{angle} = 90.09°

En la Figura 3.12.a) se muestran la posición real y estimada para el ArUco con ID 0, ubicado a una distancia considerable a la cámara. Los errores calculados para la posición del marcador fueron de 0.0283 m, 0.0146 m y 0.0723 m en los ejes x, y y z respectivamente. Por otro lado, en la Figura 3.12.b), los errores de posición del marcador ArUco de ID 1 (el cual es el más cercano a la cámara) fueron de 0.0104 m, 0.0012 m y 0.1061 m en los correspondientes ejes. Finalmente, los errores obtenidos para el ArUco de ID 2 en la Figura 3.12.c), el cual era el más alejado a la cámara, fueron de 0.0193 m, 0.0105 m y 0.0268 m en los respectivos ejes coordenados.

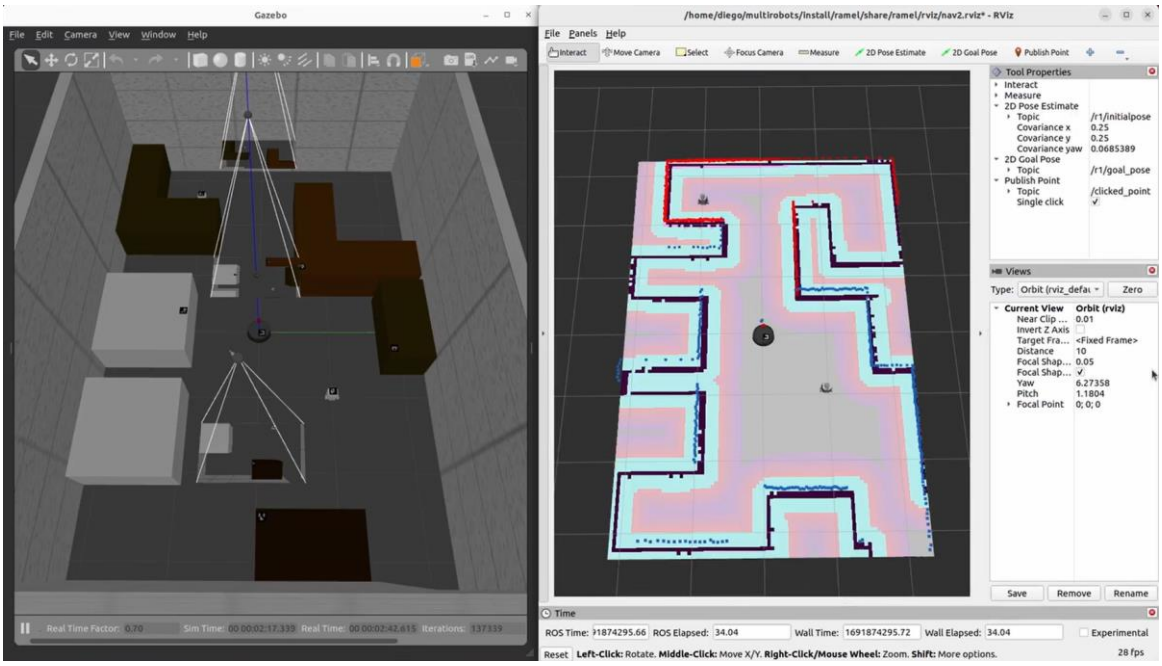
A partir de estos resultados, para el error de localización solo se considerará la posición en el plano XY, de manera que, mediante la Tabla 3.1, se obtiene un error general de posición de 0.0244 m, lo cual es un error aceptable considerando que la cámara se encontraba a una altura aproximada de 3.5 m y además las aplicaciones de movilización no requieren una precisión muy alta siempre y cuando cuente con sus sensores y módulo de detección y evasión de obstáculos. Con respecto a la orientación o rotación de los marcadores, el error promedio obtenido fue de 0.29°, un error aceptable que permite corregir problemas de orientación en el robot en caso de impactos o deslizamientos. En la aplicación a realizar, no es de suma importancia la posición en el eje z, debido a que la movilización de los robots se realizará en el plano XY, sin embargo, el error presentado en este eje es lo suficientemente bajo como para considerar utilizar esta aplicación en futuras tareas a implementar en el framework.

3.3.4 Módulo de navegación autónoma

Con el módulo de navegación autónoma implementado para n robots, cada robot tiene la capacidad de calcular la ruta óptima hacia la pose objetiva y evitar colisiones con obstáculos de manera autónoma. En la Figura 3.13 se muestra el mapa de costos local del robot 2, el cual le permite calcular la ruta óptima basándose en cuadrículas de ocupación.

Figura 3.13

Mapa de costos local



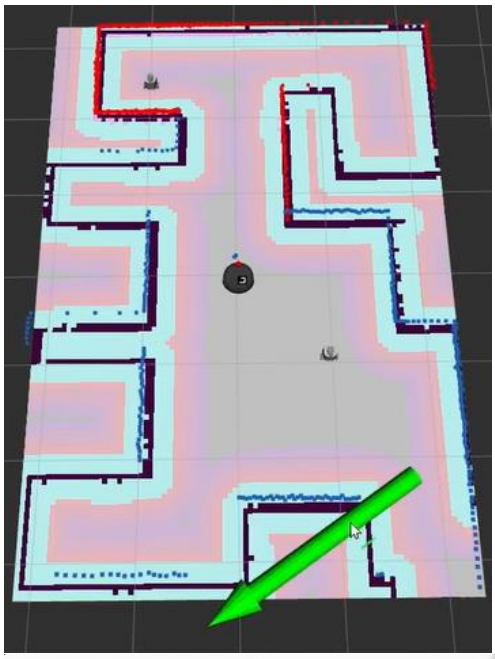
En la Figura 3.14 se muestra una secuencia de navegación de dos robots. De forma consecutiva, se les asignó una pose objetiva con las herramientas de RVIZ. Los robots calculan la ruta óptima para llegar a la pose y siguen dicha ruta hasta llegar a la pose objetiva. Las poses objetivas de cada robot se pueden diferenciar por los namespaces empleados.

En caso de encontrarse con un obstáculo, el robot recalculará la ruta óptima para llegar a la pose objetiva. En la Figura 3.15, se muestra una secuencia en la cual se agrega un cubo en el ambiente simulado de Gazebo para representar un obstáculo fortuito. Previamente, se le había asignado al TurtleBot3 (R2) una pose objetiva al otro lado del mapa. El robot empieza a seguir la ruta calculada, pero al percibir el nuevo obstáculo crea una nueva ruta y lo logra evadir para llegar a la pose asignada.

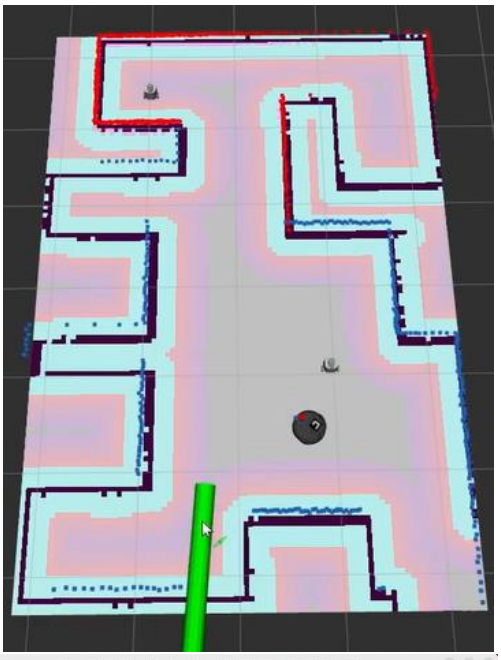
Figura 3.14

Navegación a pose objetiva de varios robots

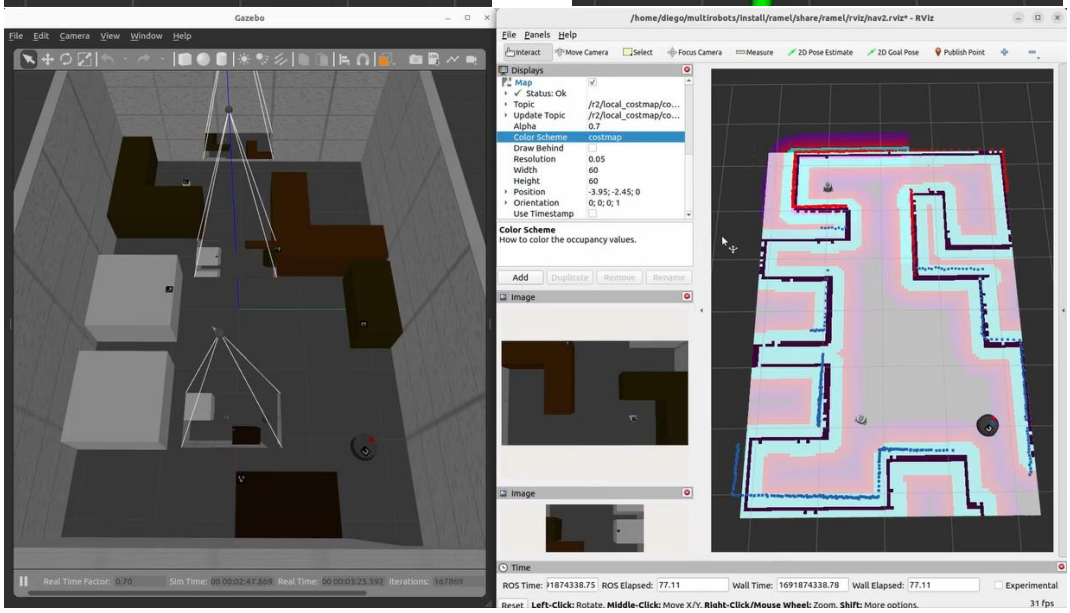
a)



b)



c)



Nota. a) Pose establecida para el Create3, b) Pose establecida para el Turtlebot, c) Robots en la pose objetiva comandada.

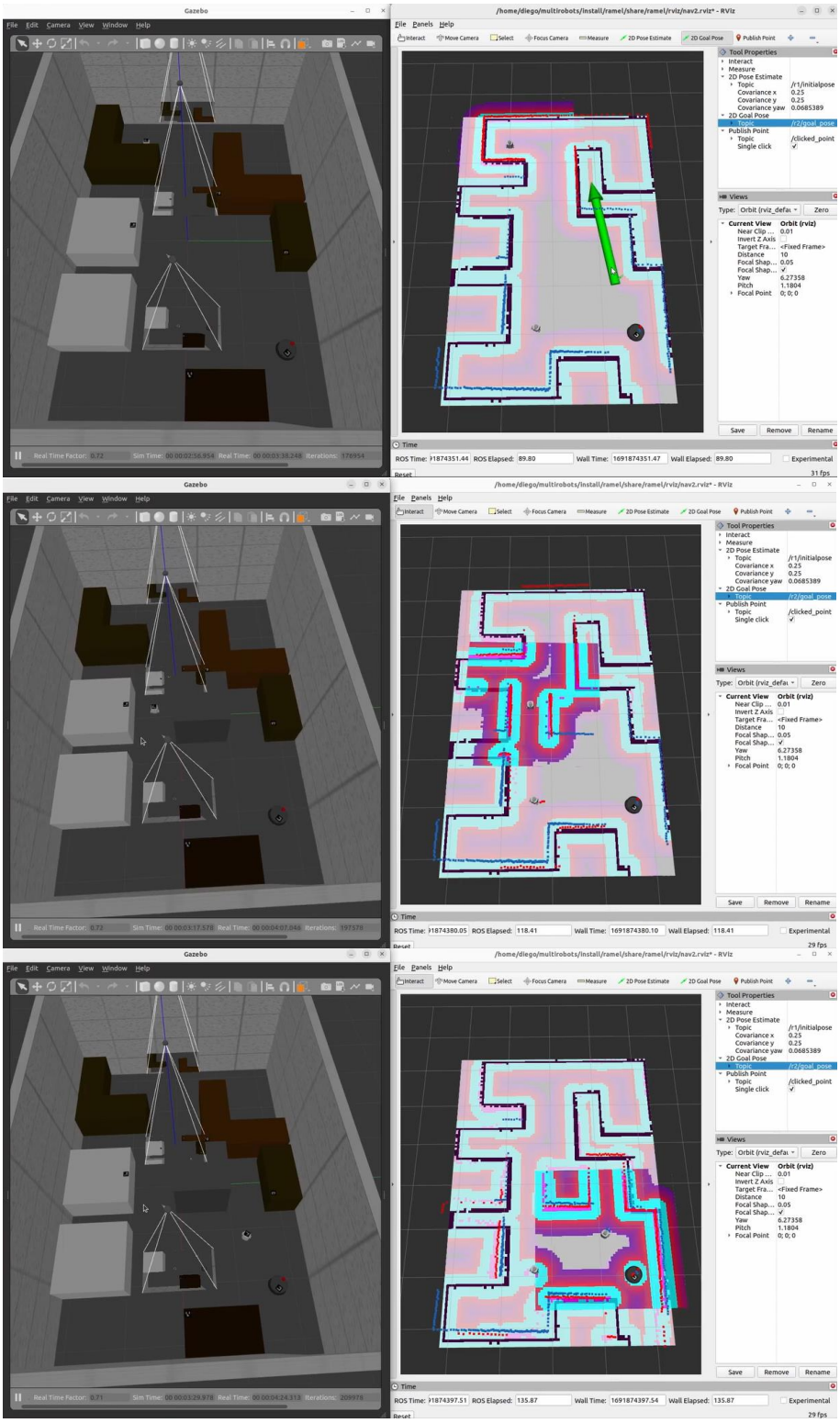
Figura 3.15

Evasión de obstáculos

a)

b)

c)



Nota. a) Petición de movimiento a un robot turtlebot3 hacia un destino, b) Evasión de un obstáculo añadido en el momento, c) Llegada al destino por parte del robot.

3.3.5 Módulo de controlador de tareas

El framework multirobot desarrollado cuenta con un mánager de tareas el cual se encarga de asignar tareas a los robots, ejecutar comandos de actualización de posición y escoger al robot que debe ejecutar la tarea en función de la distancia a la que se encuentra en el caso que no se especifique el robot por parte del operador.

En la Figura 3.16 se visualiza la ejecución de la tarea de recogida y entrega en donde se ha especificado al framework, mediante la interfaz gráfica, el requerimiento para que un robot se acerque al puesto de trabajo 0, recoja un objeto y lo entregue en el puesto de trabajo 2, en este caso la tarea es ejecutada por el robot 1.

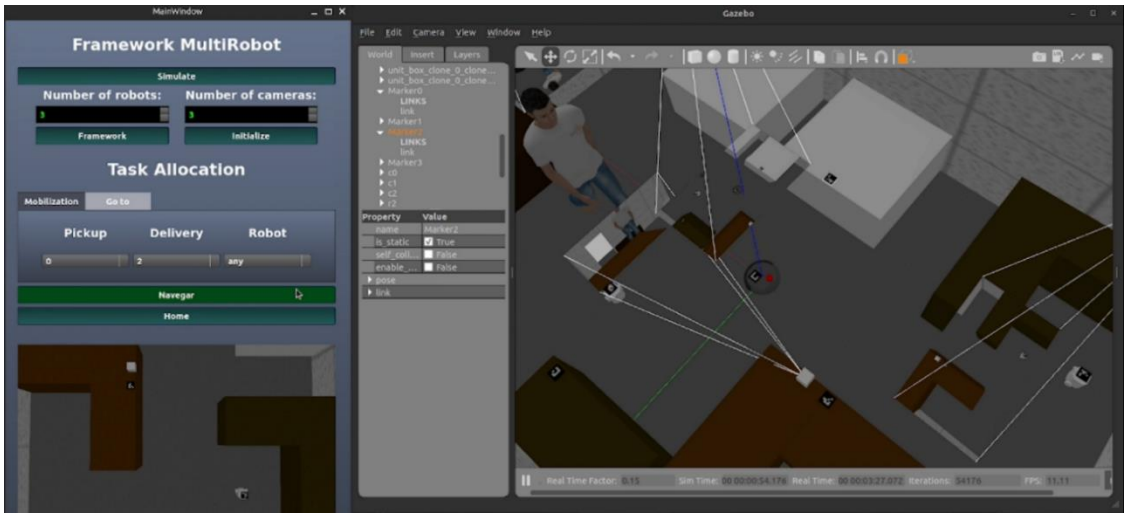
En la Figura 3.17 se visualiza la ejecución de la tarea de ir a una posición en donde se ha especificado al framework, mediante la interfaz gráfica, el requerimiento para que un robot se acerque al punto (x,y) = (-2,0), siendo esta tarea ejecutada por el robot 2.

En ambas tareas mostradas anteriormente no se especifica el robot que debe ejecutar la tarea en cuestión, por este motivo, el manager obtiene las posiciones de cada robot así como sus estados (libre u ocupado), eligiendo un robot para ejecutar la acción considerando que dicho robot sea el más próximo al punto de destino (Pickup en el caso de la tarea “Mobilization” de recogida y entrega de productos, y el punto (x,y) especificado en el caso de la tarea “Go To”. Por este motivo, en el primer caso se asignó la tarea al robot 1, mientras que en el segundo caso se asignó la tarea al robot 2 los cuales durante la asignación se encontraban cerca de los puntos de destino y libres de cualquier otra tarea.

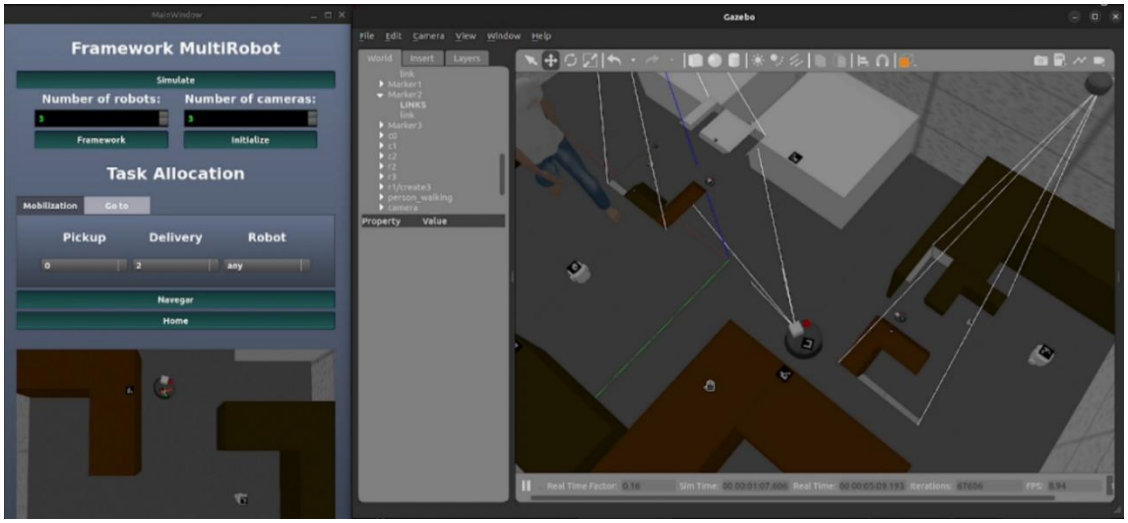
Figura 3.16

Asignación de tarea: Delivery de objetos

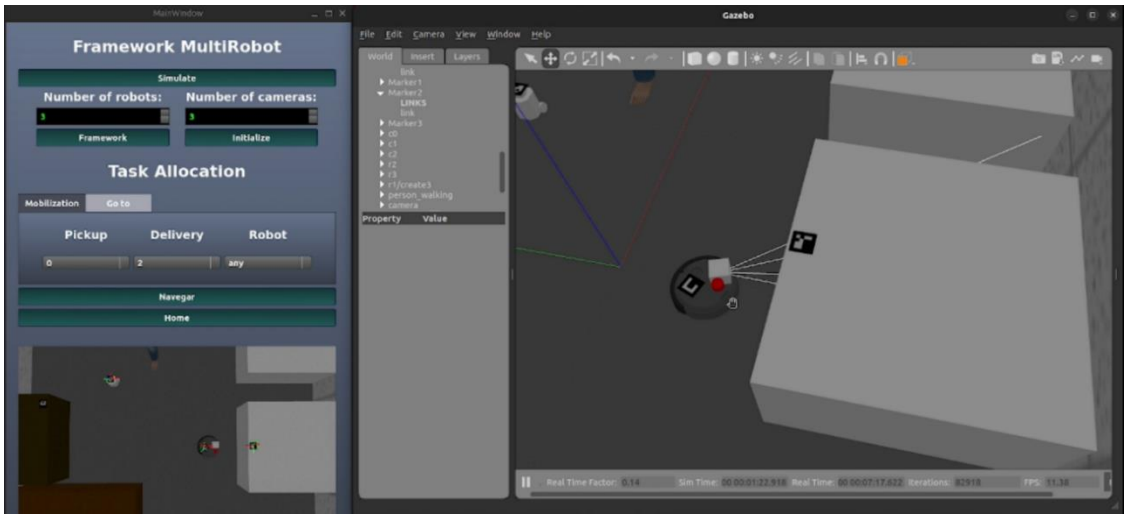
a)



b)



c)



Nota. a) Petición de la tarea a un robot cercano, b) Recogida de objeto en el puesto de trabajo 0, c) Entrega de objeto al destino puesto de trabajo 2.

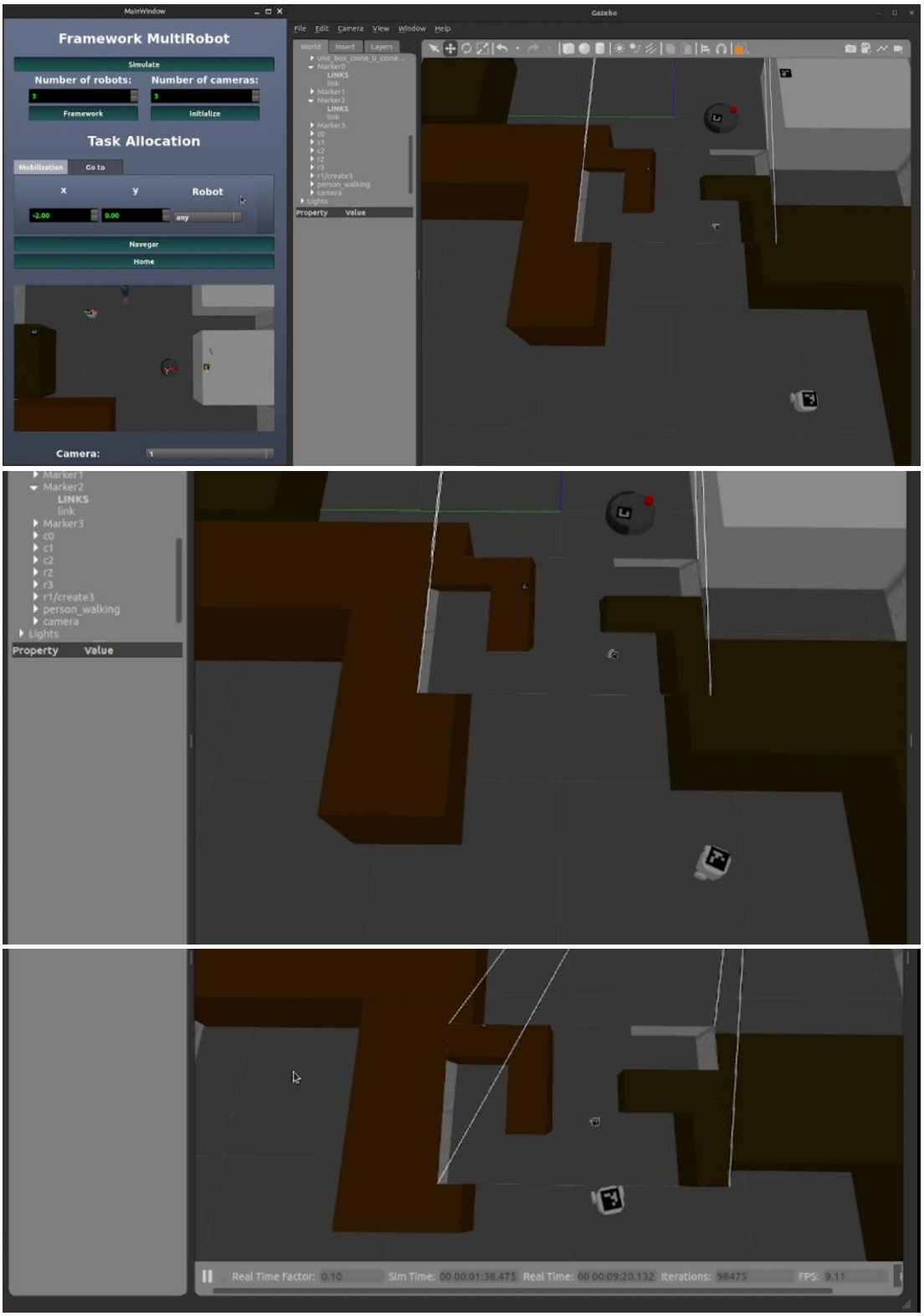
Figura 3.17

Ejecución de tarea ir a posición

a)

b)

c)



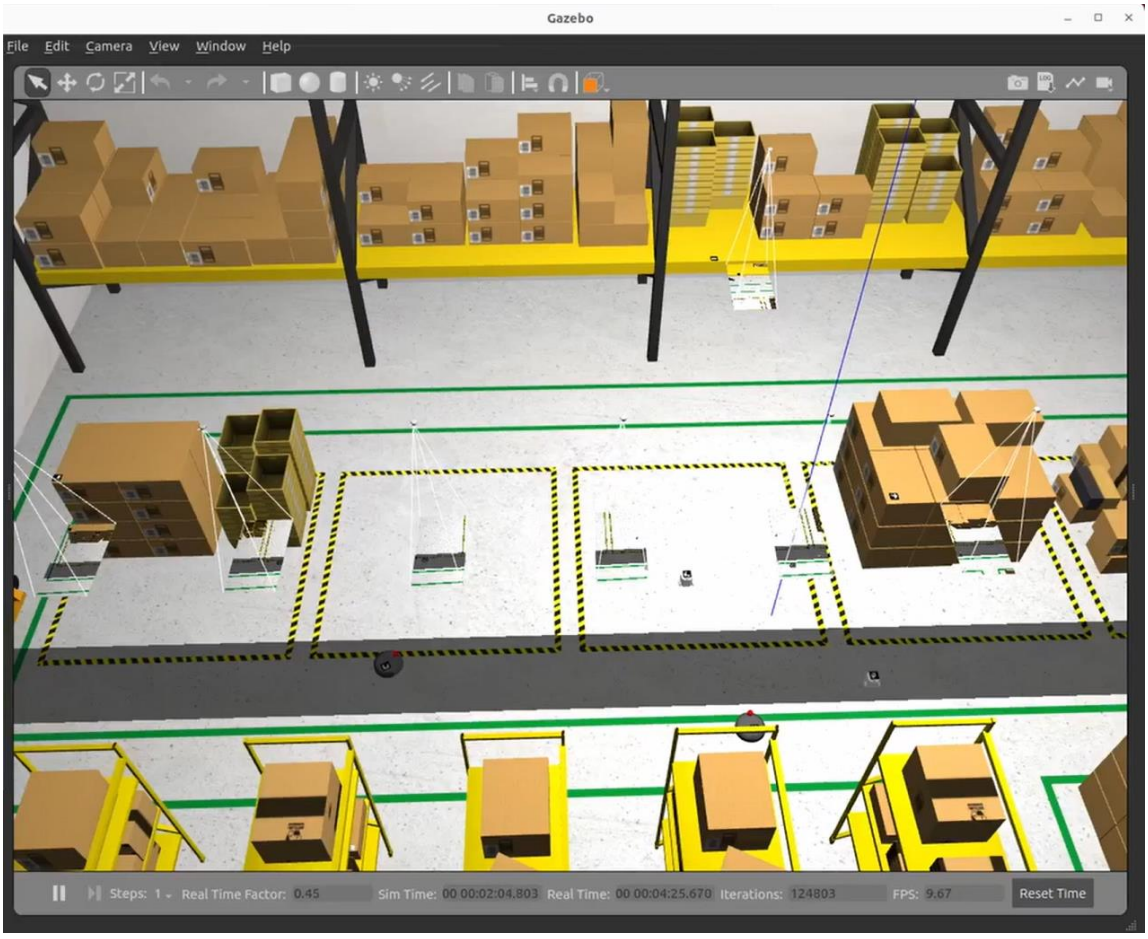
Nota. a) Petición de la tarea a un robot cercano, b) Inicio del movimiento a la posición destino, c) Llegada del robot a la posición destino.

3.3.6 SMR en diferentes ambientes simulados

Se utilizó un ambiente simulado para representar un galpón industrial y con el framework poder elaborar un SMR que realice entrega y recogida de paquetes entre diferentes secciones del galpón. Para este caso, se utilizaron 2 robots turtlebot3 y 2 robots créate, obteniendo como resultado el ambiente simulado mostrado en la Figura 3.18 ya con las cámaras y los marcadores aruco agregados. Los marcadores aruco se agregaron en 3 distintas secciones del galpón, en zonas de almacenamiento de paquetes.

Figura 3.18

Ambiente simulado galpón industrial

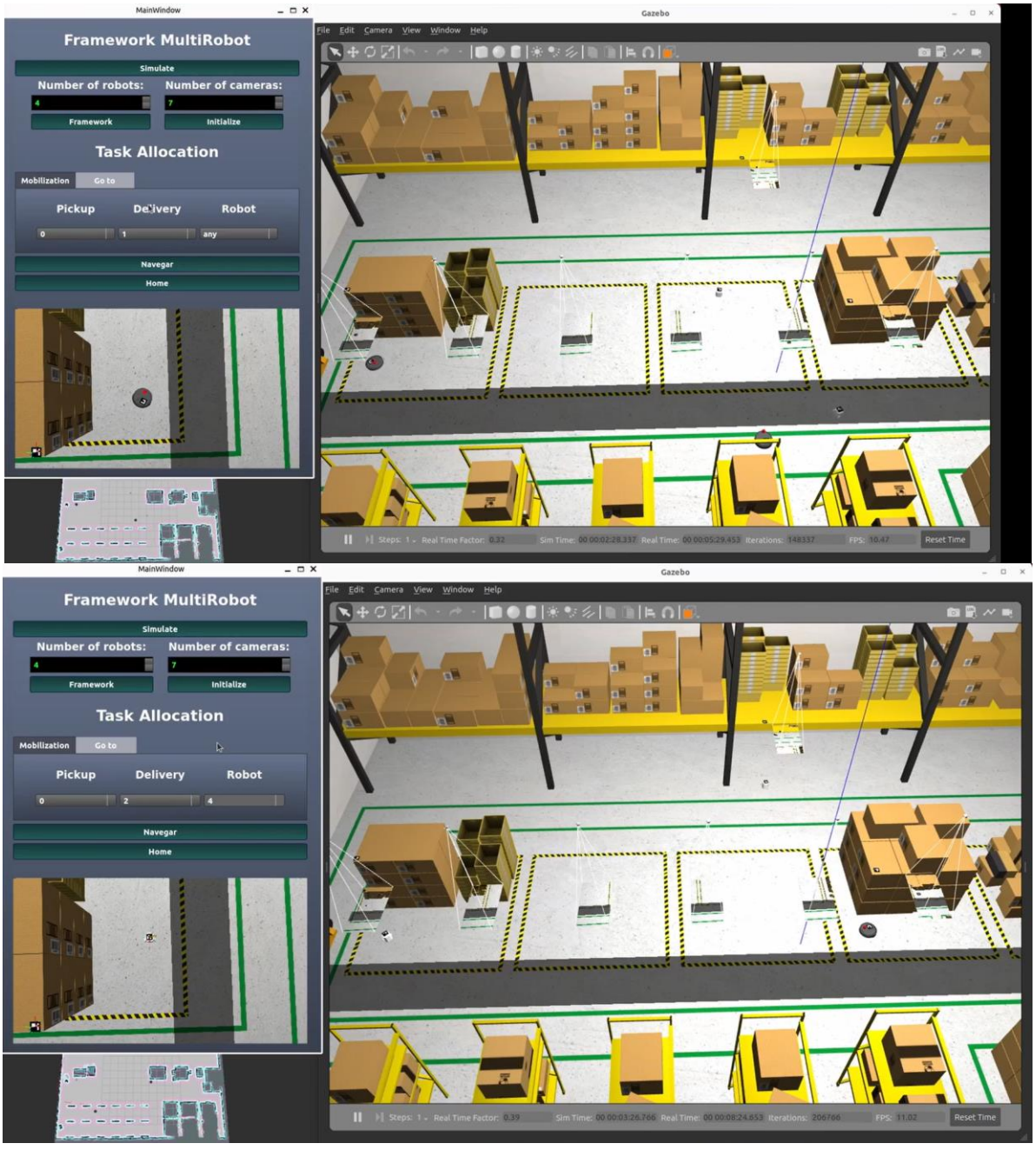


En la Figura 3.19, se muestran fotogramas de un video en el cual se comanda varias tareas de entrega y recogida de paquetes consecutivamente utilizando la interfaz gráfica de usuario. Para cada tarea comandada, en caso de que no se especifique el robot, se acerca el

robot más cercano que este desocupado. Los robots culminan sus tareas de recogida y entrega entre las estaciones del galpón y vuelven a estar disponibles para recibir nuevas tareas.

Figura 3.19

SMR en galpón industrial

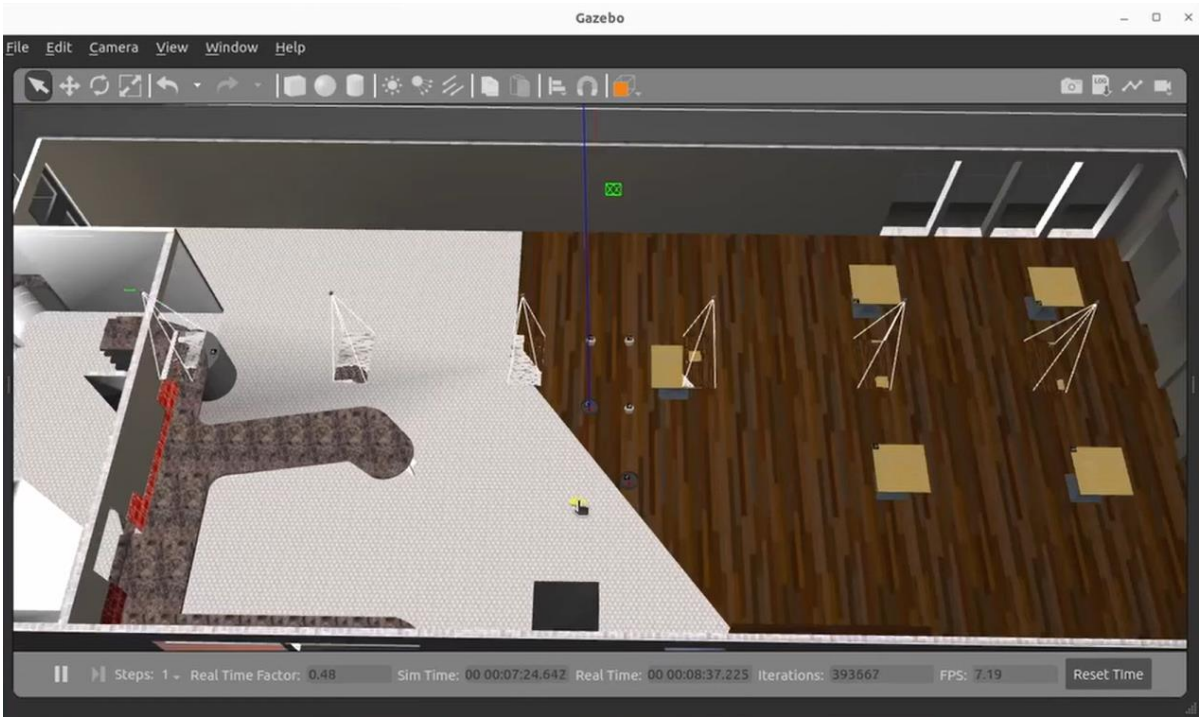


De la misma forma, se utilizó un ambiente simulado para representar un restaurante o cafetería con el objetivo de realizar tarea de recogida y entrega de vajilla y platos de comida. Se agregó un Turtlebot3, teniendo en total 5 robots en este SMR. En la Figura 3.20 se muestra

el resultado del ambiente simulado con todos los componentes necesarios para utilizar el SMR. Los marcadores aruco se ubicaron uno en cada una de las 5 mesas para los clientes. Además, se agregó un marcador aruco en el mesón contiguo a la cocina, para que los robots puedan llevar y recoger los platos de la cocina.

Figura 3.20

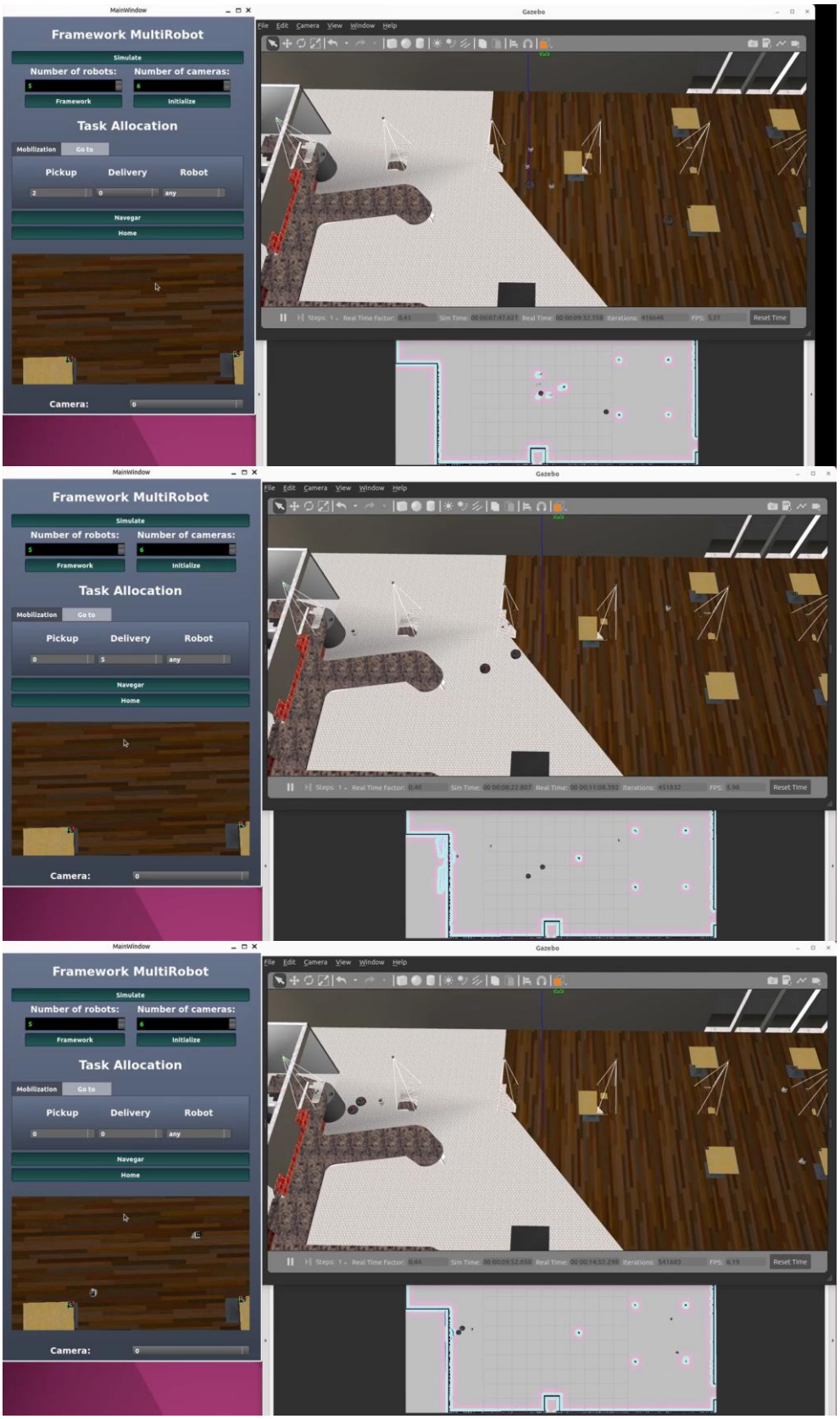
Ambiente simulado cafetería



En la Figura 3.21, se muestran secuencias de un video en el que se desarrollan varias tareas de recogida y entrega de vajilla comandadas desde la interfaz gráfica de usuario. Cabe resaltar, que la estación de recogida cercana a la cocina llega a tener por momentos mucha congestión de robots ya que es una estación fija para cada tarea. Sin embargo, los robots logran evadirse entre ellos y no se llega a efectuar ninguna colisión. Los robots terminan de ejecutar su tarea asignada y vuelven a estar disponibles para nuevas tareas comandadas.

Figura 3.21

SMR en cafetería



3.4 Análisis de costos

Para el desarrollo del Framework Multirobot se necesita de una inversión inicial que incluya el diseño y programación de los distintos componentes del framework previo al inicio del modelo de negocio planteado, el cual consiste en prestar servicios a las empresas que deseen automatizar y robotizar sus plantas de producción o de servicios, tal y como se mostró en las aplicaciones de galpón y cafetería. De esta manera se considerarán trabajos complementarios de instalación, paquetes de cámaras y robots que utilicen los usuarios, y funcionalidades extras que se requieran, por otro lado, este servicio contará con mantenimiento del framework y servicio técnico en caso de ser requerido.

En la Tabla 3.2 se enlistan los costos de inversión inicial necesarios para la puesta en marcha del negocio, mientras que en la Tabla 3.3 se desglosan los costos asociados a la empresa y sus actividades. Finalmente, en la Tabla 3.4 se enlistan los rangos de precios ofrecidos para los clientes, siendo que el precio final dependerá del tipo de aplicación, tareas a implementar, número de robots y sensores, implementos requeridos y área de trabajo.

Tabla 3.2

Costos de inversión inicial estimados

Descripción	Cantidad	Precio Unit. (USD)	Precio Total (USD)
Programadores módulos de navegación autónoma y tareas	1	$ 1500.00	$ 1500.00
Programadores módulo de visión artificial	1	$ 1500.00	$ 1500.00
Programadores interfaz gráfica	1	$ 1000.00	$ 1000.00
Computadores	3	$ 1300.00	$ 3900.00
Software	3	$ 22.00	$ 66.00
Servicios básicos (luz e internet)	1	$ 100.00	$ 100.00
TOTAL			$ 8066.00

Tabla 3.3

Desglose de costos

Descripción	Cantidad	Precio Unit. (USD)	Precio Total (USD)
Costos Fijos			**$ 1600.00**
Servicios básicos (luz e internet)	-	$ 100.00	$ 100.00
Técnicos de instalación	2	$ 500.00	$ 1000.00
Técnicos de mantenimiento	1	$ 500.00	$ 500.00
Costos Variables			**$ 350.00 /uni**
Revisión y mantenimiento (promedio)	-	$ 100.00	$ 100.00
Instalación (promedio)	-	$ 250.00	$ 250.00

Tabla 3.4

Ingresos

Descripción	Precio min. (USD)	Precio max. (USD)
Instalación	$ 1000.00	$ 5000.00
Suscripción renovación cada 6 meses (mantenimiento, ampliación, entre otros)	$ 500.00	$ 1500.00

Considerando una venta media de $ 3000 por instalación y $ 1000 por subscripción, el punto de equilibrio para cada una de las actividades ofrecidas (instalación y subscripción) se obtienen en las ecuaciones 3.6 y 3.7:

$$P.E. = \frac{CF}{P - CV} = \frac{\$\ 1600.00}{(\$\ 3000\ ) - (\$\ 250.00)} = 0.58 \times 12 \approx 7\ instalaciones\ anuales \quad (3.6)$$

$$P.E. = \frac{CF}{P - CV} = \frac{\$\ 1600.00}{(\$\ 1000) - (\ \$\ 100.00)} = 1.8 \approx 2 \times 12 = 24\ \ subscripciones\ anuales \quad (3.7)$$

De manera general, el modelo de negocio planteado no siempre realizará instalaciones de manera continua, sin embargo, las instalaciones realizadas acumularán consigo subscripciones, por este motivo se plantea el siguiente objetivo: 2.5 instalaciones anuales (5 instalaciones cada 2 años) y su correspondiente pago de 2 renovaciones de suscripción por año

por cada instalación realizada durante los primeros 5 años, utilizando las ecuaciones 3.8 y 3.9 se obtiene el siguiente balance esperado de la empresa en la Tabla 3.5.

$$Ingresos = 2.5 \times \$\,3000 + año \times instalaciones\ obj. \times 2\ renovaciones \times \$\,1000 \quad (3.8)$$

$$Egresos = 2.5 \times \$\,250 + año \times instalaciones\ obj. \times 2\ renovaciones \times \$\,100 \quad (3.9)$$

Tabla 3.5

Balance esperado del negocio

Año	0	1	2	3	4	5
Ingresos		$ 12500,00	$ 17500,00	$ 22500,00	$ 27500,00	$ 32500,00
Egresos	$ 8066,00	$ 20325,00	$ 20825,00	$ 21325,00	$ 21825,00	$ 22325,00
Flujo	$ -8066,00	$ -7825,00	$ -3325,00	$ 1175,00	$ 5675,00	$ 10175,00
Balance Total	$ -8066,00	$ -15891,00	$ -19216,00	$ -18041,00	$ -12366,00	$ -2191,00

A partir de los resultados obtenidos en la Tabla 3.5, se espera recuperar la inversión y empezar a generar ganancias netas durante el transcurso de año 6, siempre y cuando se cumplan los objetivos planteados y los costos correspondan a los descritos anteriormente, no obstante, es posible que se necesite de liquidez durante estos años para expandir el negocio.

Capítulo 4

4.1 Conclusiones y Recomendaciones

En esta sección se describen las conclusiones del presente proyecto en función de los resultados obtenidos y los objetivos planteados al inicio del documento, enfocándose en la importancia del proyecto y su posible aplicación. Además, se enlistan recomendaciones y trabajos adicionales para mejorar el diseño propuesto, aprovechar nuevas tecnologías e investigaciones en desarrollo y denotar problemas a solucionar en trabajos posteriores.

4.1.1 Conclusiones

En la industria contemporánea, identificada con una necesidad ascendente de automatización y colaboración en ambientes cerrados, el FrameWork MultiRobot desarrollado resultó ser un avance importante. Utilizando ROS2 para lograr la comunicación y coordinación eficaz de diversos sistemas mecatrónicos se logró obtener una solución robusta, flexible y escalable; parámetros claves en los procesos industriales. Las conclusiones principales del trabajo se listan a continuación:

- Se obtuvo una comunicación y coordinación efectiva entre los agentes del sistema mediante la arquitectura de software MultiRobot en ROS2. La arquitectura de software permite añadir más robots y más sensores al sistema. La robustez de la arquitectura resalta en la capacidad de respuesta de los robots en situaciones de alta complejidad, como la evasión de obstáculos.
- Se desarrolló un ambiente simulado para representar RAMEL, pudiendo diseñar progresivamente el FrameWork MultiRobot. Asimismo, se utilizaron ambientes simulados de un galpón industrial y de una cafetería para validar el funcionamiento del FrameWork; obteniendo un funcionamiento correcto para flotas de hasta 6 robots y flotas heterogéneas en los distintos entornos.
- Se consiguió localizar la posición y orientación precisa de los robots y los espacios de trabajo en el entorno con técnicas de visión por computadora mediante el uso de

marcadores ArUco. Además, módulo de visión por computadora permite inicializar las posiciones de los robots en el mapa de navegación y rectificar sus posiciones luego de culminar o fallar en una tarea asignada. El error de localización de marcadores ArUco es de ±0.025 m en el plano XY, y un error de rotación de 0.29°.

- El uso de sensores LiDAR y técnicas SLAM permitió la navegación autónoma en ambientes cerrados simulados. Gracias a esto, el agente es capaz de localizarse en el mapa mientras se encuentra en movimiento, comparando la información del LiDAR con el mapa generado. Además, el LiDAR le permite al robot identificar obstáculos no presentes en el mapa para que pueda computar una nueva ruta hacia su destino.
- Se elaboró un controlador de tareas asistido por visión artificial para poder asignar misiones de movilización de objetos al robot más cercano al punto de recogida, mejorando la eficiencia operativa. Esto permite la asignación de varias misiones en simultáneo, siendo aplicable en galpones industriales, hospitales, restaurantes, entre otras aplicaciones. Además, se agregó una misión de ir a una coordenada XY al controlador de tareas, permitiendo asignar diferentes misiones desde el mismo controlador.
- En conclusión, el trabajo desarrollado tiene un impacto significativo en el campo de la robótica colaborativa y la automatización de sistemas. A comparación con trabajos existentes, el FrameWork MultiRobot brinda una solución integral que abarca la desde percepción mediante visión artificial para disminuir la incertidumbre en el posicionamiento de los agentes, hasta la asignación y ejecución de tareas colaborativas. Los sistemas MultiRobot implementados con el FrameWork desarrollado tendrán como fortalezas factores como la robustez, escalabilidad, adaptabilidad y flexibilidad.

4.1.2 Recomendaciones

En el presente proyecto se realizaron algunas consideraciones para facilitar el proceso de diseño, sin embargo, para dotar de mayor robustez al sistema se considera trabajar en aspectos como:

- Incorporar el control de vehículos aéreos no tripulados y brazos robóticos en el FrameWork MultiRobot, para poder realizar misiones más complejas, en especial de manipuladores, integrando las diversas tareas que pueden llegar a ejecutar los agentes dentro del mánager de tareas de framework.
- Integrar técnicas de aprendizaje automático para mejorar la navegación y toma de decisiones de los robots, así como también para la diferenciación, mediante redes convolucionales, entre los diversos tipos de robots que conforman la flota, de manera que el operador pueda requerir la asistencia de un tipo específico de robot sin necesidad de conocer su id.
- Integrar paqueterías para el uso de sensores LiDAR 3D para que los robots puedan tener una mejor percepción del entorno, reduciendo aún más el riesgo de colisiones y permitiéndoles realizar un mayor número de tareas con seguridad.
- Emplear Visual SLAM, como alternativa del uso sensores LiDAR 3D, para el mapeo y la navegación autónoma de los agentes, integrando los paquetes correspondientes en caso de que los robots cuenten con modelos de cámaras que permitan realizar este tipo de mapeo.
- Se recomienda añadir métodos de localización para los robots con el fin de brindar mayor robustez al sistema, una de las formas es usar la información de odometría del propio robot y verificarla mediante visión artificial como en el presente proyecto, esto debido a que las cámaras no siempre podrán captar la posición del robot.

- Las cámaras del módulo de visión artificial necesitan estar conectadas al máster publicando su información de calibración e imagen mediante protocolos de ROS2, por lo que se recomienda aumentar una opción para poder utilizar cámaras que no cuenten con paquetes de ROS2, ya sea que se obtenga la imagen de manera inalámbrica mediante la red local de internet, o que la información de calibración se obtenga de un archivo que deba proporcionar el usuario.

Referencias

[1] M. Mena Roa, “El número de robots industriales se ha triplicado en la última década. Automatización Industrial.”, oct. 2022. Consultado: el 26 de agosto de 2023. [En línea]. Disponible en: https://es.statista.com/grafico/28534/stock-operativo-mundial-de-robots-industriales

[2] A. Gautam y S. Mohan, “A review of research in multi-robot systems”, en *2012 IEEE 7th International Conference on Industrial and Information Systems, ICIIS 2012*, 2012. doi: 10.1109/ICIInfS.2012.6304778.

[3] T. Balch y L. Parker, *Robot Teams: from diversity to polymorphism.* . 2002.

[4] T. Lochmatter y A. Martinoli, “Understanding the Potential Impact of Multiple Robots in Odor Source Localization”, ene. 2009.

[5] J. J. Roldán-Gómez, J. De León Rivas, P. Garcia-Aunon, y A. Barrientos, “Una revisión de los sistemas multi-robot: Desafíos actuales para los operadores y nuevos desarrollos de interfaces”, *RIAI - Revista Iberoamericana de Automatica e Informatica Industrial*, vol. 17, núm. 3, pp. 294–305, 2020, doi: 10.4995/RIAI.2020.13100.

[6] I. Fassi, S. Pio Negri, C. Pagano, L. Rebaioli, y M. Valori, “Robots Industriales 4.0”, en *Fabricación digital para pymes*, J. C. Chaplin, C. Pagano, y S. Fort, Eds., 2020, pp. 209–231.

[7] M. Sachon, “Los pilares de la industria 4.0”, *Revista de Negocios del IEEM*, pp. 46–54, abril de 2018.

[8] ISO, “ISO 8373:2021(en) Robotics — Vocabulary”, 2021. https://www.iso.org/obp/ui/#iso:std:iso:8373:en (consultado el 15 de junio de 2023).

[9] J. Cuautle, E. Berra, y M. Pérez, “Robótica cooperativa para el transporte de objetos”, 2018.

[10] P. U. Lima y L. M. Custódio, “Multi-Robot Systems”, Lisboa, 2005. [En línea]. Disponible en: www.springerlink.com

[11] L. Zhang, Y. Sun, A. Barth, y O. Ma, “Decentralized control of multi-robot system in cooperative object transportation using deep reinforcement learning”, *IEEE Access*, vol. 8, pp. 184109–184119, 2020, doi: 10.1109/ACCESS.2020.3025287.

[12] W. Shule, C. M. Almansa, J. P. Queralta, Z. Zou, y T. Westerlund, “UWB-Based Localization for Multi-UAV Systems and Collaborative Heterogeneous Multi-Robot Systems”, en *Procedia Computer Science*, Elsevier B.V., 2020, pp. 357–364. doi: 10.1016/j.procs.2020.07.051.

[13] S. Mutawe, M. Hayajneh, S. Banihani, y M. Al Qaderi, “Simulation of Trajectory Tracking and Motion Coordination for Heterogeneous Multi-Robots System”, *Jordan Journal of Mechanical and Industrial Engineering*, vol. 15, núm. 4, pp. 337–345, oct. 2021, [En línea]. Disponible en: https://www.researchgate.net/publication/355796473

[14] F. Matoui, B. Boussaid, B. Metoui, y M. N. Abdelkrim, “Contribution to the path planning of a multi-robot system: centralized architecture”, *Intell Serv Robot*, vol. 13, núm. 1, pp. 147–158, ene. 2020, doi: 10.1007/s11370-019-00302-w.

[15] Á. Madridano, A. Al-Kaff, D. Martín, y A. de la Escalera, “Trajectory planning for multi-robot systems: Methods and applications”, *Expert Syst Appl*, vol. 173, jul. 2021, doi: 10.1016/j.eswa.2021.114660.

[16] E. Klavins, “Communication Complexity of Multi-robot Systems”, en *Algorithmic Foundations of Robotics V*, J. D. Boissonnat, J. Burdick, K. Goldberg, y S. Hutchinson, Eds., 2004, pp. 275–291.

[17] K. C. Chen, S. C. Lin, J. H. Hsiao, C. H. Liu, A. F. Molisch, y G. P. Fettweis, “Wireless Networked Multirobot Systems in Smart Factories”, *Proceedings of the IEEE*, vol. 109, núm. 4, pp. 468–494, abr. 2021, doi: 10.1109/JPROC.2020.3033753.

[18] H. Durrant-Whyte y T. Bailey, “Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms”, 2006.

[19] M. Kalaitzakis, B. Cain, S. Carroll, A. Ambrosi, C. Whitehead, y N. Vitzilaios, “Fiducial Markers for Pose Estimation: Overview, Applications and Experimental Comparison of the ARTag, AprilTag, ArUco and STag Markers”, *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 101, núm. 4, abr. 2021, doi: 10.1007/s10846-020-01307-9.

[20] D. A. Cantero-Alonso y E. A. Martínez-Jara, “Visión por computadora: identificación, clasificación y seguimiento de objetos”, 2014.

[21] R. Muñoz-Salinas, M. J. Marín-Jimenez, E. Yeguas-Bolivar, y R. Medina-Carnicer, “Mapping and localization from planar markers”, *Pattern Recognit*, vol. 73, pp. 158–171, ene. 2018, doi: 10.1016/j.patcog.2017.08.010.

[22] OpenCV, “Detection of ArUco Markers”. https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html (consultado el 17 de julio de 2023).

[23] S. Macenski, F. Martín, R. White, y J. G. Clavero, “The Marathon 2: A Navigation System”, feb. 2020, doi: 10.1109/IROS45743.2020.9341207.

[24] Open Navigation LLC, “Nav2”, 2020. https://navigation.ros.org/index.html (consultado el 17 de julio de 2023).

Apéndices

Apéndice A

Localización de los indicadores ArUco

La librería utilizada de ArUco permite obtener la posición de un indicador con respecto a la cámara que lo está visualizando, no obstante, para su implementación en un sistema multirobot fue necesario incluir una referencia en común para todos los indicadores, para lo cual se utilizan otras funciones extras de desarrollo propio las cuales, mediante el uso de matrices de transformación homogénea, permite obtener la posición de los indicadores con respecto a la nueva referencia.

A partir de la librería utilizada, al momento de realizar la detección de los marcadores se obtiene un vector de posición como el de la ecuación A.1, y un vector de rotación como el de la ecuación A.2.

$$p = (x_c, y_c, z_c) \tag{A.1}$$

$$r = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix} \tag{A.2}$$

Esta información es suficiente para poder definir la matriz de transformación homogénea que indica la posición y orientación del indicador con respecto a la cámara que lo detecta. Para esto se calcula el ángulo de la matriz de rotación que corresponde a la norma del vector de rotación, tal y como se muestra en la ecuación A.3. seguido se calcula el vector de rotación unitario en la ecuación A.4 para, mediante la fórmula de Rodrigues en la ecuación A.5 proceder a calcular una matriz 3×3 de rotación del identificador con respecto a la cámara.

$$\theta = norm(r) \tag{A.3}$$

$$r = \frac{r}{\theta} = \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix} \tag{A.4}$$

$$R_i^C = \cos(\theta) + (1 - \cos(\theta)) r r^T + \sin(\theta) \begin{pmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{pmatrix} \tag{A.5}$$

De esta manera, la matriz de trasformación homogénea del identificador con respecto a la cámara está dado por la ecuación A.6.

$$H_i^C = \begin{pmatrix} R & P \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} & x_i \\ R_i^C & y_i \\ & z_i \\ 0 & 1 \end{pmatrix} \tag{A.6}$$

Por otro lado, se define una nueva matriz de transformación homogénea, esta vez de la cámara con respecto al punto de origen establecido, por lo que se expresa como en la ecuación A.7 y depende de la posición y orientación con la que el usuario instale la cámara.

$$H_C^O = \begin{pmatrix} & x_c \\ R_C^O & y_c \\ & z_c \\ 0 & 1 \end{pmatrix} \tag{A.7}$$

Finalmente, se puede calcular la posición y orientación del identificador con respecto al punto de origen mediante la ecuación A.8.

$$H_i^O = H_C^O \times H_i^C \tag{A.8}$$

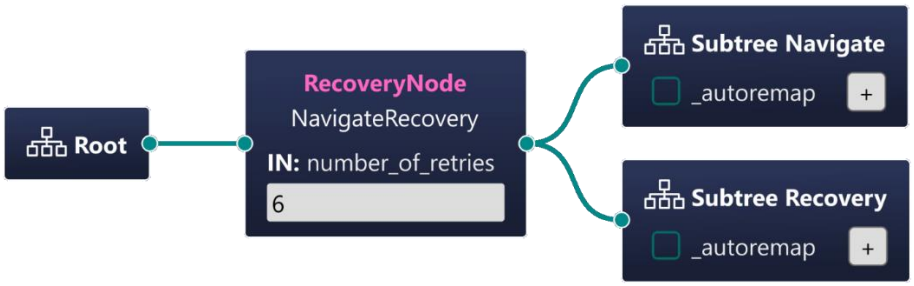
Apéndice B

Árbol de comportamientos implementados

El BT consta de dos principales subárboles que se muestran en la Figura B.1, uno de navegación y otro de recuperación. El nodo NavigateRecovery se encarga de establecer el número de intentos que el robot realizará para recuperarse de un error en el subárbol de navegación antes de pasar al subárbol de recuperación. En este caso, luego de que el subárbol de navegación retorne error 6 veces, se procederá al árbol de navegación.

Figura B.1

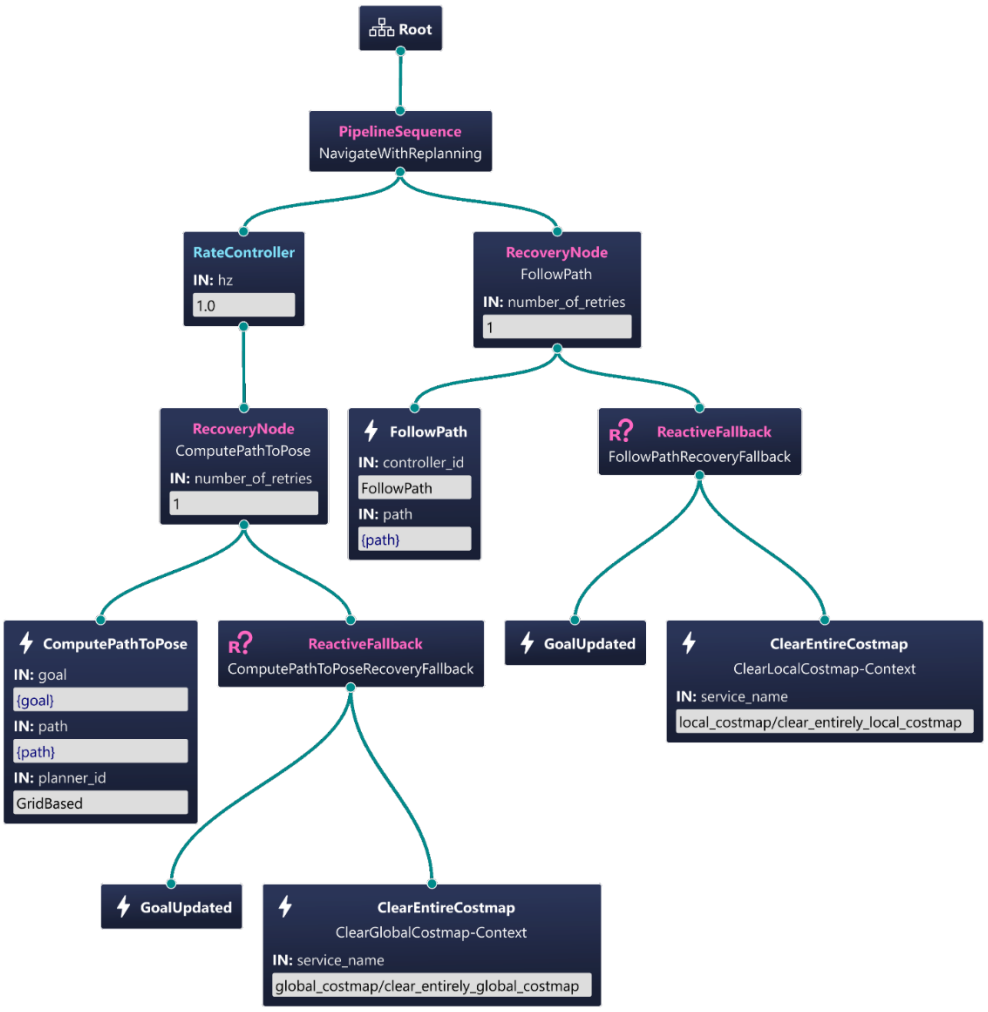
Árbol de comportamientos



El subárbol de navegación, mostrado en la Figura B.2, es el encargado de las funciones de navegación principales. Realiza secuencialmente las acciones de calcular un camino hacia el objetivo y seguir el camino. En el caso de la acción de planificación, se estableció una frecuencia de 1 Hz para prevenir sobrecargas. Si cualquiera de las acciones falla se intenta realizar recuperaciones contextuales antes de dirigirse hacia el subárbol de recuperación. Las recuperaciones contextuales para la acción de planificación son revisar si el objetivo ha sido modificado y limpiar el mapa de costo global. Por otro lado, las recuperaciones contextuales de la acción de seguir el camino son verificar cambios en el objetivo y limpiar el mapa de costos local.

Figura B.2

Subárbol de navegación

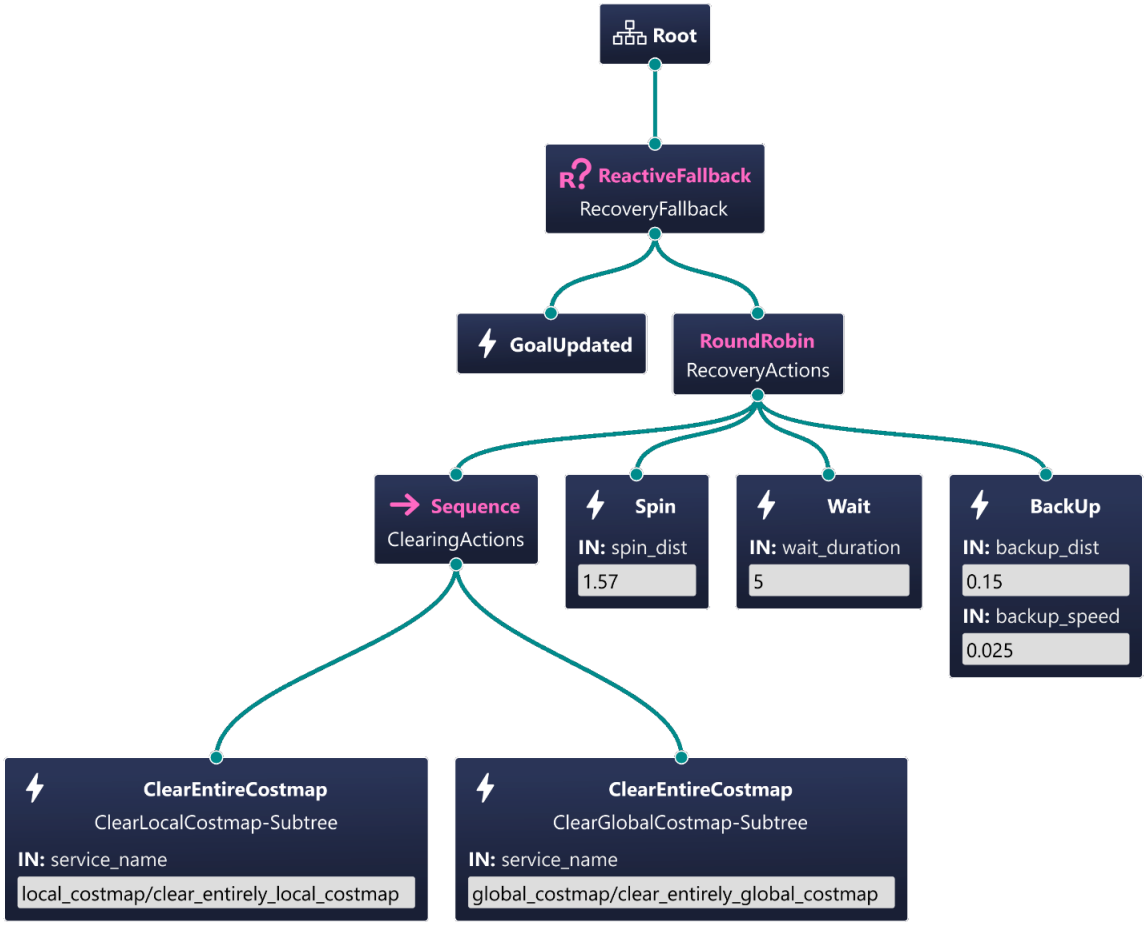


El subárbol de recuperación, mostrado en la Figura B.3, se activa cuando el árbol de navegación retorna fracaso. Este subárbol se encarga de realizar recuperaciones de nivel de sistema. El nodo de ReactiveFallback se utiliza para que en caso de que el objetivo sea actualizado, termine las acciones de recuperación y retorne éxito. Las acciones de recuperación se realizan de forma secuencial hasta que una de ellas retorne éxito o todas fracasen. Primero, se limpian los mapas de costo globales y locales. Posteriormente, el robot realiza una rotación. Luego, el robot realiza una pausa de cinco segundos. Finalmente, el robot retrocede una distancia especificada a la velocidad establecida. Si una de las acciones de recuperación funciona, el robot vuelve a intentar navegar en el subárbol de navegación. En resumen, la

estructura del árbol de navegación es robusta y flexible, permite al robot navegar de forma eficiente y manejar distintos inconvenientes.

Figura B.3

Subárbol de recuperación



Apéndice C

Archivo README.md de repositorio de GitHub

MultiRobot System Framework

POWERED BY N | Solid

Introduction

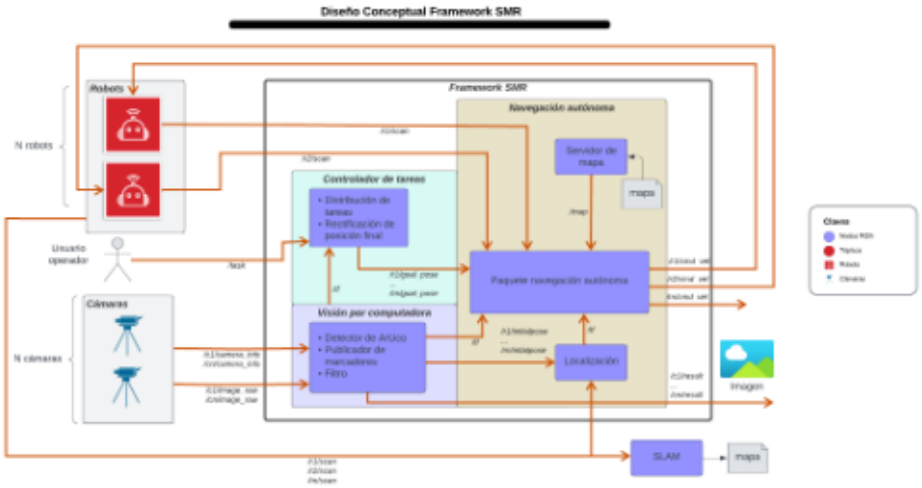
MRS Framework is a is a comprehensive platform (built by Anthonny Piguave and Diego Ronquillo) designed to facilitate the development and deployment of multirobot systems, providing a set of essential modules and tools to enable various capabilities. The framework consists of the following core components:

- Autonomous navigation module for N robots.
- Computer vision module for aruco markers detection to provide localization and rectification.
- Task manager module for distributing pick-up/delivery tasks, go to pose tasks and go to home task.
- 3 simulation environments equiped with robotic models such as Turtlebot3 and Create3 for testing.
- Graphical User Interface (GUI).

The MRS Framework is designed to be versatile, enabling researchers and developers to experiment with different robot types, test various algorithms, and build complex multirobot systems that can handle diverse tasks in real-world scenarios. By providing these core modules and simulation environments, the MRS Framework accelerates the development and deployment of multirobot applications while fostering collaboration and innovation in the field of robotics.

Framework Overview

An overview of the framework modules and communications is presented below:



Requirements:

- Operating System: Ubuntu Linux Jammy Jellyfish (22.04)
- Consider according to the number of robots the computational requirements of the framework.

Dependencies:

- ROS2 Humble. Installation: https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html
- Verify gazebo installation (If it is not installed: https://classic.gazebosim.org/tutorials?tut=install_ubuntu)
- Verify the acquisition of the gazebo controls library. If it is not installed:

```
sudo apt-get install ros-humble-gazebo-ros2-control
```

Installation

1.- Clone the github repository in a workspace source folder. (ws_name/src)

```
git clone https://github.com/RAMEL-ESPOL/MultiRobot.git
```

2.- Go to workspace directory and Install dependencies

```
cd ..
rosdep install -y --from-paths src --ignore-src
```

3.- Build the pacakge.

```
colcon build
```

Usage

0.- Navigate to the workspace directory (The previous step to perform in all new cmd windows opened)

```
source install/setup.bash
```

1.- Launch the simulation. (In case you are working with a real system, you can skip this step)

```
ros2 launch mrs_master mrs_simulation.launch.py
```

2.- Open a new cmd window, source in the ws and launch the MRS.

```
ros2 launch mrs_master MRS.launch.py n_robots:=3 n_cams:=3
```

Where n_robots and n_cams are the numbers of robots and cameras of the system, by default 3 in both. 3.- Open a new cmd window, source in the ws and set the initial pose of the robots.

```
python3 src/MultiRobot/mrs_master/scripts/initial_pose_publisher.py n_robots
```

Be sure to replace "n_robots" with the number of robots. For example, in the case of 3 robots:

```
python3 src/MultiRobot/mrs_master/scripts/initial_pose_publisher.py 3
```

4.- Launch the task manager. Markdown is a lightweight markup language based on the formatting conventions

```
python3 src/MultiRobot/mrs_master/scripts/mrs_manager.py n_robots
```

Now you can send task commands to the robots with this format:

> `Params for navigation: <id_task> <arg1> <arg2> <id_robot>(optional).`

Where id_task is 1 for pick-up/delivery tasks and 2 for go to pose task. If the id_tak is 1, arg1 and arg2 are the id of the workstations, else if id_task if 2, these are the x and y coordinates. id_robot is the number of the robot in its namespace, for example 1 for r1 robot, if no robot is specified, the manager will choose the nearest and unoccupied robot. For instance, if we want the nearest robot to pick-up a box in station 1 and leave it in station 3, the command would be:

Where id_task is 1 for pick-up/delivery tasks and 2 for go to pose task. If the id_tak is 1, arg1 and arg2 are the id of the workstations, else if id_task if 2, these are the x and y coordinates. id_robot is the number of the robot in its namespace, for example 1 for r1 robot, if no robot is specified, the manager will choose the nearest and unoccupied robot. For instance, if we want the nearest robot to pick-up a box in station 1 and leave it in station 3, the command would be:

```
1 1 3
```

Graphical User Interface

For installing the GUI desktop aplication follow this setps.

0.- Navigate to workspace directory.

1.- Navigate to MultiRobot/mrs_master/scripts directory.

```
cd src/MultiRobot/mrs_master/scripts
```

2.- Open framework.desktop and modify paths

> Exec=/path_to_ws/src/MultiRobot/mrs_master/scripts/framework.sh

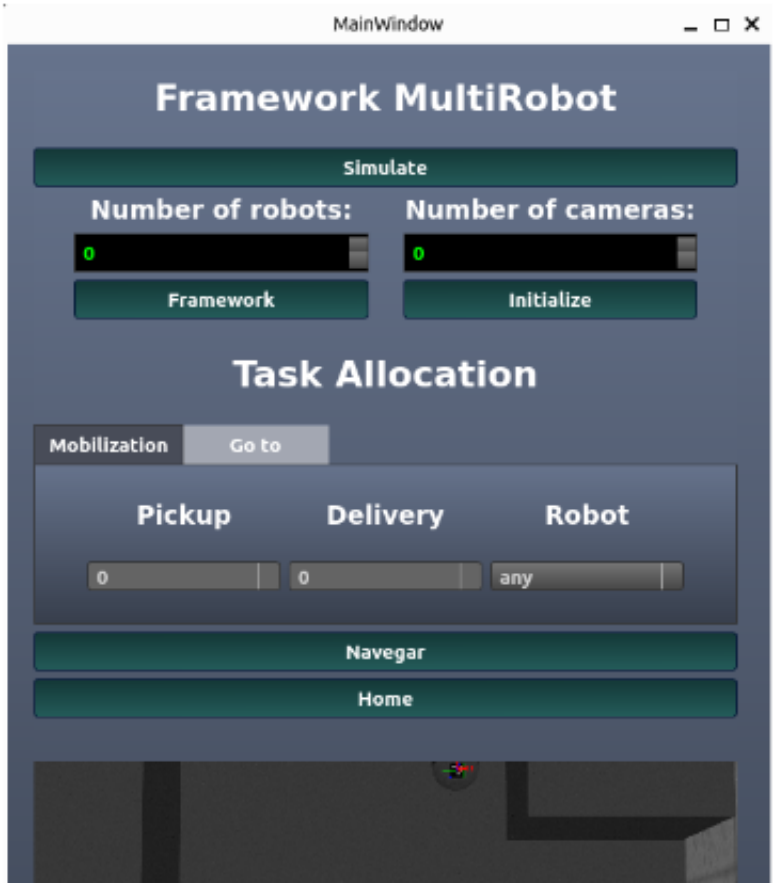
> Icon=/path_to_ws/src/MultiRobot/mrs_images/icon.png

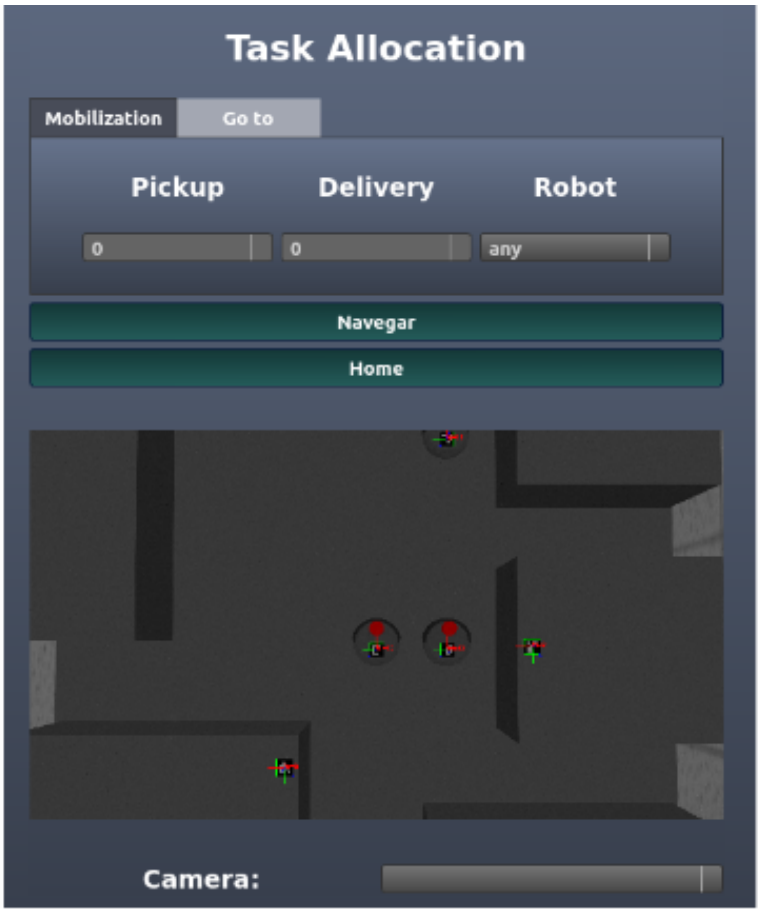
3.- Move the GUI application to Desktop

```
mv framework.desktop ~/Desktop/
```

4.- Right click the icon and choose "Allow launching"

Now you are able to open the GUI from Desktop.

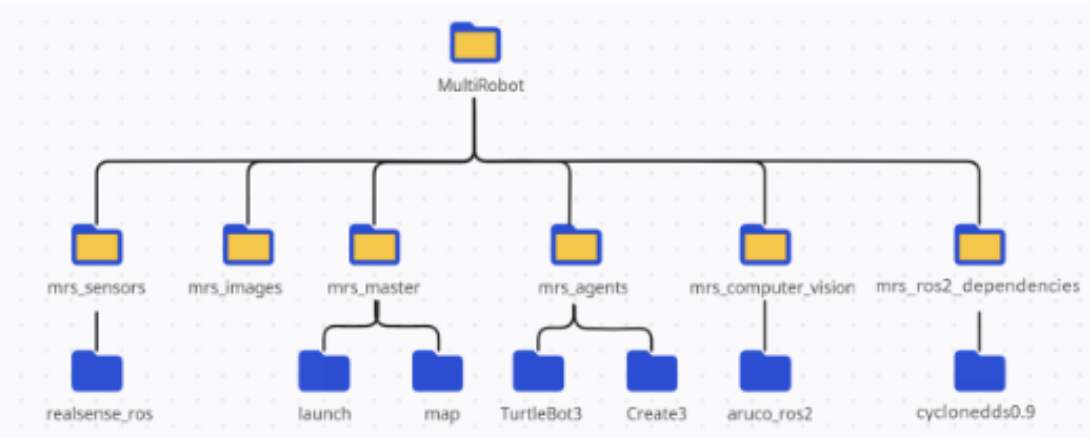




Package Structure

The package structure is the following:

- mrs_agents -> includes the robots' packages (In this repo iRobot Create3 and Turtlebot3 robots are included)
- mrs_computer_vision -> includes the aruco libraries (Library, interfaces and nodes)
- mrs_images -> includes images of the project.
- mrs_master -> contains the framework, task master and GUI
- mrs_sensors -> contains cameras' and other sensors' packages (In this repo Realsense L515 camera is included)
- mrs_ros2_dependencies -> contains cyclonedds configuration (v0.9 able to sim multi-robots)



Apéndice D

Repositorio en Github