**Escuela Superior Politécnica del Litoral**

**Facultad de Ingeniería en Mecánica y Ciencias de la Producción**

# Diseño de una arquitectura de software para el control maestro-esclavo de múltiples robots

INGE-2620

**Proyecto Integrador**

Previo la obtención del Título de:

**Ingeniero en Mecatrónica**

Presentado por:

Iesus René Dávila Aguilar

César Patricio Quintuña Llivichuzca

Guayaquil - Ecuador

Año: 2024

# Dedicatoria

El presente proyecto es dedicado a Dios, mis padres, hermanos, mi sobrina y amigos, quienes me han ayudado en toda mi carrera universitaria tanto los momentos buenos o malos, dándome apoyo siempre que necesitaba.

**Iesus Dávila**

Este proyecto lo dedico a Dios, mis padres, mi hermano y amigos quienes fueron mi soporte cuando quise rendirme y me han ayudado a seguir con este camino llamado vida.

**Cesar Quintuña**

# Agradecimientos

Agradezco a Dios, mi familia y amigos por siempre apoyarme.

A mi amigo, Cesar Quintuña, por dedicarle tiempo y esfuerzo al proyecto para que salga acorde a las metas planteadas.

A RAMEL, por ser un lugar de mucho aprendizaje en mis últimos años de carrera, para prepararme tanto personalmente como profesionalmente.

**Iesus Dávila**

Agradezco a Dios, a mis padres por creer siempre en mí incluso cuando fallaba.

A todo el grupo de Lávate las manos que fue lo mejor que me paso en la universidad, y en especial, a mi amigo Iesus, le agradezco su paciencia y su forma de ser espero su éxito profesional y personal, a RAMEL por darme nuevas experiencias en mi carrera

sentido a la universidad.

**Cesar Quintuña**

# Declaración Expresa

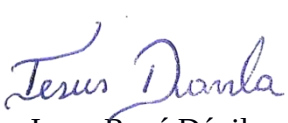
Nosotros Iesus René Dávila Aguilar y César Patricio Quintuña Llivichuzca acordamos y reconocemos que:

La titularidad de los derechos patrimoniales de autor (derechos de autor) del proyecto de graduación corresponderá al autor o autores, sin perjuicio de lo cual la ESPOL recibe en este acto una licencia gratuita de plazo indefinido para el uso no comercial y comercial de la obra con facultad de sublicenciar, incluyendo la autorización para su divulgación, así como para la creación y uso de obras derivadas. En el caso de usos comerciales se respetará el porcentaje de participación en beneficios que corresponda a favor del autor o autores.

La titularidad total y exclusiva sobre los derechos patrimoniales de patente de invención, modelo de utilidad, diseño industrial, secreto industrial, software o información no divulgada que corresponda o pueda corresponder respecto de cualquier investigación, desarrollo tecnológico o invención realizada por mí/nosotros durante el desarrollo del proyecto de graduación, pertenecerán de forma total, exclusiva e indivisible a la ESPOL, sin perjuicio del porcentaje que me/nos corresponda de los beneficios económicos que la ESPOL reciba por la explotación de mi/nuestra innovación, de ser el caso.

En los casos donde la Oficina de Transferencia de Resultados de Investigación (OTRI) de la ESPOL comunique los autores que existe una innovación potencialmente patentable sobre los resultados del proyecto de graduación, no se realizará publicación o divulgación alguna, sin la autorización expresa y previa de la ESPOL.

Guayaquil, 13 de mayo del 2024.



Iesus René Dávila Aguilar

César Patricio Quintuña Llivichuzca

# Evaluadores

Firmado electrónicamente por:
BRYAN JOAO
PURUNCAJAS MAZA

**Bryan Puruncajas, Ph.D.**

Profesor de Materia

Firmado electrónicamente por:
FRANCISCO XAVIER
YUMBLA AREVALO

**Francisco Yumbla, Ph.D.**

Tutor de proyecto

**Resumen**

En el campo de los sistemas multirobots los UGVs destacan por su capacidad de navegación autónoma, pero existen tareas que, debido a limitaciones individuales, no pueden ser completadas eficazmente. Este proyecto tiene como objetivo desarrollar una arquitectura maestro-esclavo en un sistema multirobot, utilizando ROS2 Humble, para abordar tareas de manera colaborativa. La hipótesis plantea que una estructura jerárquica mejora la coordinación y la eficiencia en entornos complejos, optimizando el sistema y su fiabilidad. La relevancia del proyecto radica en la creciente demanda de sistemas autónomos colaborativos en sectores industriales y de exploración. El desarrollo incluyó el uso de UGVs y ROS2 Humble, aplicando técnicas de programación orientada a objetos y patrones de diseño robustos. Se respetaron normas de seguridad y estándares de software, realizando pruebas exhaustivas en entornos simulados. Los resultados confirmaron que, bajo la estructura maestro-esclavo, el sistema completa todas las tareas asignadas, incluso ante fallos de uno o varios robots. Además, se diseñó una aplicación de escritorio intuitiva, que facilita la operación del sistema sin necesidad de conocimientos especializados en ROS2. En conclusión, se demuestra que la arquitectura propuesta es eficaz para gestionar y delegar tareas en sistemas multirobots terrestres, estableciendo una base sólida para futuras investigaciones.

**Palabras Clave:** Arquitectura de Software, ROS2, Multirobots, Maestro-Esclavo

***Abstract***

*In the field of multi-robot systems, UGVs stand out for their autonomous navigation capabilities, but there are tasks that, due to individual limitations, cannot be completed effectively. This project aims to develop a master-slave architecture in a multi-robot system, using ROS2 Humble, to tackle tasks collaboratively. The hypothesis is that a hierarchical structure improves coordination and efficiency in complex environments, optimizing the system and its reliability. The relevance of the project lies in the growing demand for collaborative autonomous systems in industrial and exploration sectors. The development included the use of UGVs and ROS2 Humble, applying object-oriented programming techniques and robust design patterns. Safety regulations and software standards were respected, carrying out exhaustive tests in simulated environments. The results confirmed that, under the master-slave structure, the system completes all assigned tasks, even in the event of failure of one or more robots. In addition, an intuitive desktop application was designed, which facilitates the operation of the system without the need for specialized knowledge in ROS2. In conclusion, the proposed architecture is shown to be effective for managing and delegating tasks in terrestrial multirobot systems, establishing a solid foundation for future research.*

*Keywords: Software Architecture, ROS2, Multirobots, Master-Slave*

## Índice general

## Abreviaturas

| | |
|---|---|
| ESPOL | Escuela Superior Politécnica del Litoral |
| IESS | Instituto Ecuatoriano de Seguridad Social |
| MRS | Multi Robot System |
| RAMEL | Robotics Automation & Mechatronics Engineering Laboratory |
| ROS | Robot Operating System |
| ROS2 | Robot Operating System 2 |
| RViz | Robot Visualization |
| UAV | Unmanned Autonomous Vehicle |
| UGV | Unmanned Ground Vehicle |
| YAML | Ain't Markup Language |

## Índice de figuras

## Índice de tablas

# Capítulo 1

## 1.1 Introducción

Las industrias a lo largo de los años han sufrido varios cambios que han significado una mejora en los procesos que se llevan a cabo, incorporando nuevas tecnologías para automatizar procesos o tareas. En el reporte del 2023, realizado por International Federation of Robotics llamado "World Robotics 2023 Report: Asia ahead of Europe and the Americas" [1] indican que hubo una cifra récord de 553,052 instalaciones de robots industriales en fábricas mundiales, obteniendo un incremento del 5% en comparación al año 2022.

Un problema en las industrias son las regulaciones gubernamentales de cada país, las cuales son exigentes para evitar que las personas trabajen en condiciones donde se pone en riesgos la parte física o mental de los trabajadores, el IESS [2] posee una tabla donde indica el peso máximo que puede cargar una persona de acuerdo a su edad. Por ejemplo, los varones de 18 a 21 años pueden soportar una carga máxima de 25 libras.

En vista a las limitaciones gubernamentales, las industrias han migrado a soluciones robóticas impulsando un incremento en la elaboración e investigación de robots móviles y articulados. En el último reporte de Allied Market Research titulado "Industrial Robotics Market Size, Share, Competitive Landscape and Trend Analysis Report by Type, by End user industry, by Function: Global Opportunity Analysis and Industry Forecast, 2023-2032" [3], se menciona que en el 2020 se estimó que el valor del mercado mundial de robots industriales era de 38 mil millones de dólares, y se proyecta que aumente a 163 mil millones para 2032, con un crecimiento anual compuesto del 12.6% previsto entre 2023 y 2032.

En vista de todos los datos planteados de la robótica, cada vez es más común poder visualizar robots industriales realizando trabajos de carga pesada, exploración de lugares desconocidos o

trabajos en condiciones extremas. Además, las industrias están migrando a soluciones que engloben en sus planes actividades de robótica.

## 1.2 Descripción del problema

La mayoría de las cosas en el universo funcionan de forma jerárquica, la responsabilidad de tener un líder es muy importante porque es el encargado de que todos cumplan con su trabajo establecido. El orden jerárquico en la toma de decisiones para los robots es crucial, ya que pueden fallar al igual que muchas otras cosas, los robots pueden fallar por diferentes razones desde problemas con sus sensores para control como los actuadores para efectuar cierta acción [4]. Tener un robot maestro que sea capaz de revisar que todos cumplan con su trabajo facilita la supervisión de tareas en lugares donde el acceso humano es peligroso o complejo.

En la actualidad, existen diferentes tipos de robots móviles que permiten realizar diferentes tareas usando paquetes con gran soporte por la comunidad como Nav2 [5], sin embargo, una limitante común es que funcionan de manera independiente, y en caso de tener varios robots no existe una gestión directa entre ellos. El problema mencionado no es solo navegar varios robots de manera continua, sino tener un sistema capaz de enviar y recibir órdenes de manera jerárquica.

La limitante en la cantidad de robots que se pueden controlar en un sistema multirobot empleando una metodología maestro-esclavo es el recurso de la computadora principal. Estos elementos no son perfectos y los cálculos matemáticos, se limitan a los recursos computacionales y memoria que posee cada robot [6].

El proyecto se fundamenta en las necesidades de Robotics Automation & Mechatronics Engineering Laboratory (RAMEL) [7], dicho laboratorio tiene sede en la Escuela Superior Politécnica del Litoral (ESPOL) [8], donde se trabaja en diferentes proyectos de investigación. RAMEL requiere una estructura de gestión de las tareas realizadas por los diferentes robots autónomos terrestres para realizar tareas de navegación dentro de un entorno cerrado.

## 1.3 Justificación del problema

Desarrollar una solución al problema descrito conlleva diferentes áreas de un ambiente mecatrónico, como el control, navegación de robots, programación de aplicaciones complejas y creación de sistemas inteligentes. El desarrollo de un sistema robusto, escalable y sostenible permite que la solución perdure a lo largo de los años.

En los últimos años, la investigación de sistemas multirobot cada vez tiene un mayor impacto en el mundo de la robótica, sin embargo, estos sistemas se basan simplemente en la ejecución y trabajo de varios robots sin un orden o una estructura especifica de navegación. El control de todos los robots es crucial para un trabajo efectivo, no es lo mismo decenas de robots funcionando sin un orden correspondiente que decenas de robots con un funcionamiento ordenado donde toda tarea respeta la posición jerárquica dentro del sistema.

Abordar una solución a este problema no solo significaría un avance tecnológico, sino la iniciativa a la creación de estándares que sean un pilar fundamental para la creación de robots. Esto facilitaría la ejecución de múltiples robots en tiempo real aplicando una metodología maestro-esclavo o investigando nuevas variantes de control para múltiples robots.

## 1.4 Objetivos

### 1.4.1 Objetivo general

Diseñar una metodología maestro-esclavo dentro de un sistema multirobot que permita realizar tareas de manera conjunta, siguiendo patrones de orden jerárquico usando ROS 2 Humble.

### 1.4.2 Objetivos específicos

1. Desarrollar un sistema multirobot escalable que permita el correcto control de los robots, usando la metodología de maestro-esclavo.

2. Elaborar en ROS2 Humble una arquitectura de software para ejecutar el sistema, consumiendo la menor cantidad de recursos y distribuyendo de manera efectiva las tareas a realizar.
3. Desarrollar una interfaz gráfica que permita la configuración, visualización y monitoreo del sistema completo en tiempo real, empleando una comunicación directa con ROS 2.

## 1.5 Marco teórico

En esta sección se abordan los conceptos necesarios para proporcionar una visión general del proyecto, su campo de aplicación y los desafíos que enfrentan cada una de las áreas involucradas.

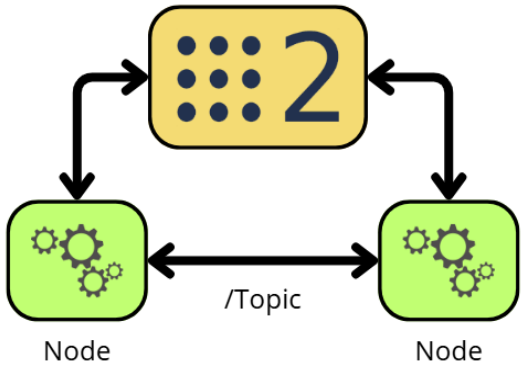
### 1.5.1 Robot Operating System (ROS)

ROS es un entorno en el cual se puede desarrollar y programar múltiples funcionalidades para un robot en específico, facilitando así el acceso a diversas bibliotecas que ha desarrollado la comunidad a lo largo del tiempo. Estas bibliotecas contienen la lógica necesaria para la mayoría de las operaciones que realizan los robots.

Según la página oficial de ROS [9], existe alta compatibilidad debido a la arquitectura y estructura distribuida, por lo que la lógica se ejecuta en procesos denominados Nodos, denotado en la Figura 1.1, permitiendo la personalización del código y que se ejecute junto a otras tareas sin presentar inconvenientes en su funcionamiento.

**Figura 1.1**

*Arquitectura de ROS2*




Los lenguajes de programación en los cuales se codifican estos procesos son C++ y Python de forma predominante, aunque también existen implementaciones en prueba en Java, Rust e Interfaces de Programación de Aplicaciones (APIs) en JavaScript. ROS tiene una trayectoria en el campo de la robótica por lo que ha ido sacando versiones y actualmente se encuentra en ROS2, framework en el cual se realizara la arquitectura propuesta.

### 1.5.2 ROS2 en las smart factories

El desarrollo continuo de la tecnología ha hecho que cada vez más robots ingresen a la industria debido a los menores costos de operación y que realizan las tareas de forma más "inteligente" siendo ROS un elemento fundamental en el desarrollo futuro de la robótica según la Asociación Española de Robótica y Automatización [10], dando paso a distribuciones como ROS2 y ROS-Industrial que se basa en aplicaciones enfocadas directamente en la industria.

Estas adaptaciones surgen basado en la necesidad de comunicar los diversos dispositivos base de la industria 4.0 y continua con la implementación de la robótica tanto para uso industrial como su expansión a áreas como la salud, asistencia que ya implican el uso de la industria 5.0 centrada en el ser humano, la sostenibilidad y la resiliencia según Forbes [11].

El uso de estas tecnologías permite la automatización de los robots en la industria, permitiendo un control especifico que garantiza el funcionamiento óptimo, y abre la posibilidad de investigaciones de diversas variantes de control de los robots en las líneas de producción.

### 1.5.3 Sistemas multirobot

Los MRS (Multi Robot System) consisten en dos o más robots autónomos trabajando de forma colaborativa, es decir, trabajan en conjunto, mostrado en la Figura 1.2, para completar tareas consumiendo menos tiempo y realizando operaciones más complejas adaptándose así al entorno que lo rodea aprovechando la sinergia de los robots [12].

**Figura 1.2**

*Robots terrestres trabajando en conjunto [13]*

Además, se caracterizan por ser eficientes, eficaces, versátiles y tolerantes a fallos haciéndolos una alternativa más factible respecto al trabajo de un solo robot [14], estas características se vuelven fundamentales a medida que los requerimientos de una tarea aumentan. Este tipo de sistemas abarcan un amplio campo de investigación que se relaciona con sectores como la inteligencia artificial, sistemas multiagente y robótica de control, siendo así que su

implementación adapta este sistema a las tareas, sin embargo, todos estos avances vienen acompañados de retos como la programación y la gestión de interferencias de tareas en los robots que aún siguen en investigación y mejora [12].

### 1.5.4 Teoría de control

Los sistemas de control en la robótica son los encargados de definir el comportamiento del robot para realizar tareas específicas, como el movimiento de los motores de las ruedas que permite su desplazamiento o la reacción ante ciertos patrones detectados por la cámara del robot, siendo allí donde radica su importancia puesto que, siempre el robot interactúa con algo mientras se encuentra realizando una tarea. Sin embargo, esta lógica de control no se cierra únicamente en un solo individuo, ya que los sistemas de control pueden englobar un entorno en el que una orden implicaría la acción de múltiples robots, siempre manteniendo la premisa de que una variable externa el sistema provee una respuesta. Este tipo de sistemas fundamentalmente buscan ser estables asegurando una resistencia ante perturbaciones y errores siguiendo criterios definidos haciendo fácil de implementar y operar bajo un procesador en la cual se ejecutará el controlador permitiendo una supervisión en tiempo real [15].

### 1.5.5 Sistema maestro-esclavo

La arquitectura de un robot comprende la presencia de sensores, actuadores, estructura, alimentación y unidad de control, que permiten al robot interactuar con el entorno mediante la interpretación de las magnitudes físicas en señales eléctricas que serán dirigidos por la unidad de control, explicada en la sección 1.5.4.

Es en esta unidad de control donde se despliega el sistema maestro-esclavo, el cual, según Real Academia de Ingeniería define como control de estructura jerarquizada, este sistema se compone de dos jerarquías, la menor denominada esclavo que realiza tareas de control mientras que el de mayor jerarquía, realiza tareas de control y/o supervisión [16].

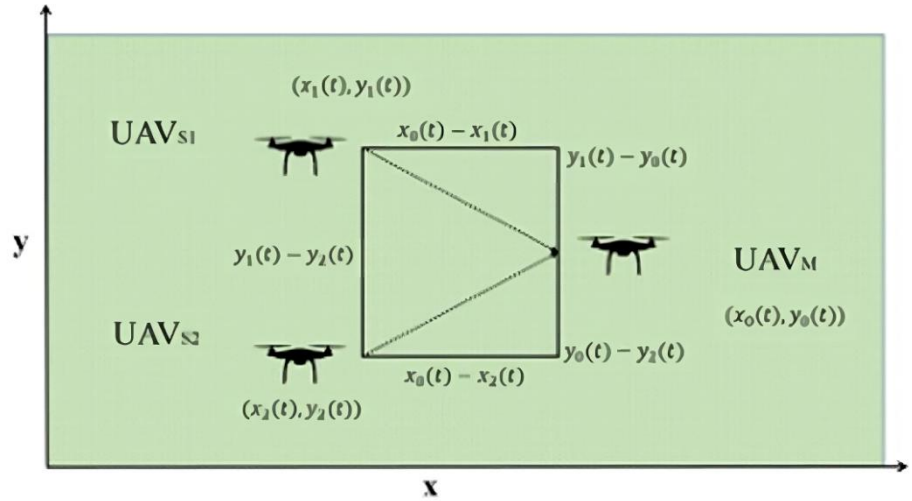
## 1.6 Estado del arte

Los avances tecnológicos han sido el pilar fundamental para que la investigación acerca de los multirobots experimente un desarrollo exponencial en los últimos años, un ejemplo de ello es Open Robot Middleware Framework (Open-RMF) que ofrece un entorno en el cual se pueden desplegar robots de diferentes fabricantes en un espacio cerrado garantizando la navegación de cada uno de ellos, y se encarga de asignar tareas a los mismos, se caracteriza por ser ampliamente modificable y permitir la operación correcta de los robots [17], constituye una forma de representar las múltiples interacciones de los robots en un entorno ya que, se presentan problemas entre robots de diferentes fabricantes brindando Open-RMF un conjunto de reglas y convenciones con el fin de evitar interrupciones en el flujo de tareas y procesos. Actualmente, dentro de las tareas disponibles se encuentran: tareas de limpieza, delivery y carga; con posibilidad a personalizar tareas por el usuario lo cual aún sigue en fase experimental. [18]

Así en el campo de la robótica de enjambre está la investigación desarrollada en UAV para establecer un modelo de comunicación usando el aprendizaje automático en operaciones de búsqueda y rescate, en la cual se define una configuración maestro-esclavo para la comunicación entre los miembros del enjambre en base a la forma de un triángulo equilátero, los drones situados en la zona izquierda corresponden a los esclavos y el dron de la derecha el maestro, garantizando que los datos y la intensidad de las señal no sufra perdidas, cada una de las posiciones se muestran con coordenadas en la Figura 1.3, mitigando las interferencias de radio frecuencia mediante el aprendizaje automático. Esta investigación aborda la predicción de la intensidad de la señal recibida (RSS) y la pérdida de potencia mediante regresión de bosque aleatorio, y describe la representación matemática de la matriz de canal [19].

**Figura 1.3**

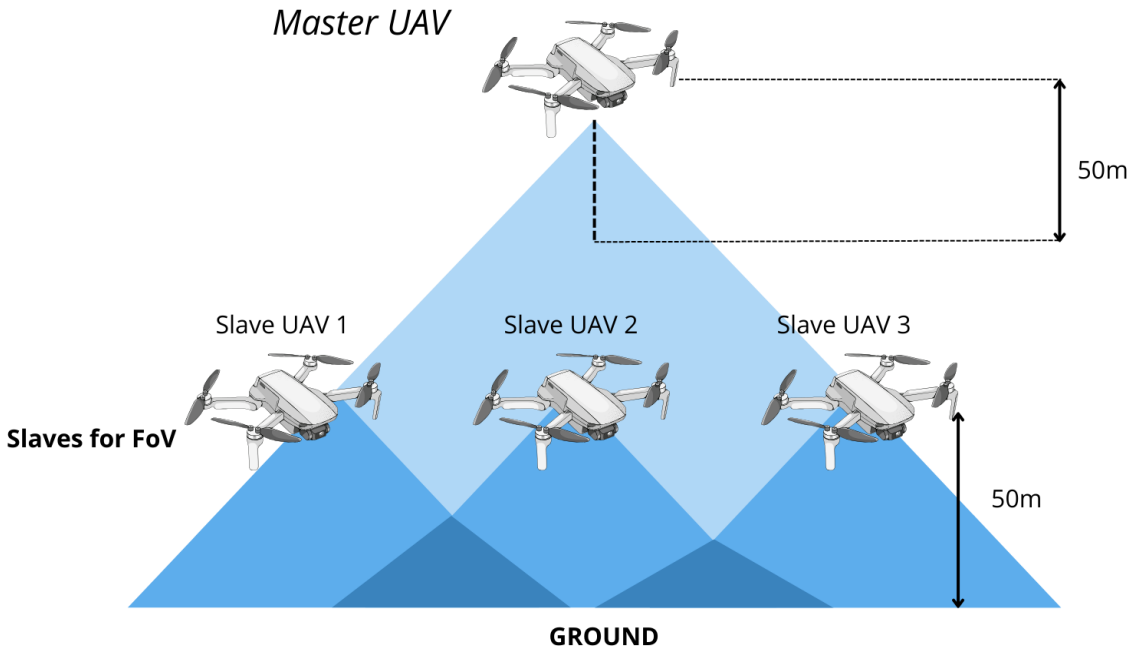
*Grafica de comunicación de los robots maestro-esclavo [19]*




Mientras que, investigaciones previas donde se incluyen un UAV maestro y múltiples UAVs esclavos comparten las tareas de toma de imágenes en la que el UAV maestro proporciona imágenes globales con un amplio FoV, en la Figura 1.4 se observan a detalle las areas de toma de imágenes y acción de los robots, mientras que los UAVs esclavos capturan imágenes locales de alta resolución reduciendo el número de UAVs necesarios para formar una matriz de cámaras con una base ancha y dinámica dando mejoras en la adquisición de imágenes capturadas por los UAVs maestro y esclavo [20].

**Figura 1.4**

*Muestra de captura de imágenes mediante la arquitectura maestro-esclavo.*



# Capítulo 2

## 2.1 Metodología.

En esta sección se describe el proceso de diseño de la arquitectura para gestión de tareas desde la formulación de posibles alternativas de solución hasta un proceso detallado de implementación conceptual, software, y control. Este subcapítulo sirve como guía para comprender el flujo de trabajo del sistema de gestión y las herramientas asociadas al producto.

## 2.2 Identificación del problema

El problema identificado se centró en la falta de un sistema capaz de controlar, delegar y ejecutar las tareas en un MRS que ya posee el laboratorio de investigación y robótica RAMEL utilizado para el despliegue de UGV, en la búsqueda de obtener un control más preciso sobre cada uno de los robots desplegados y monitorear los datos, abriendo así la posibilidad a nuevos campos de investigación asociadas a los MRS.

## 2.3 Análisis de requerimientos

Para abordar el problema de la falta de gestión de tareas en el MRS del laboratorio RAMEL fue esencial realizar un análisis exhaustivo, con el fin de desarrollar la solución más optima enfocada en mantener un equilibrio con las tecnologías usadas; siendo explicadas a continuación de forma más detallada en la Tabla 2.1.

**Tabla 2.1**

*Requerimientos del cliente*

| CLIENTE:<br>Laboratorio RAMEL | PRODUCTO:<br>Arquitectura de gestión de tareas en un MRS. |
|---|---|
| Especificaciones | |
| Concepto | Descripción |
| Gestión | Brindar un sistema automático o semiautomático para la creación, control y ejecución de tareas en un sistema multirobot. |
| Vida útil y mantenimiento | Utilizar software que reciba constantes actualizaciones a lo largo del tiempo, además que sea compatible con las herramientas tecnológicas usadas en el laboratorio. |

| | |
|---|---|
| Accesibilidad | Las configuraciones de las tareas y las rutinas deben ser claras y fáciles de editar para el usuario. |
| Adaptabilidad | El sistema debe ser capaz de integrar robots terrestres con diferentes tipos de control (diferenciales, omnidireccionales), además de poder integrar múltiples subsistemas dentro del sistema. |
| Monitoreo en tiempo real | El sistema debe brindar información acerca del estado de la cada una de las tareas de los robots una vez se haya ejecutado en el sistema. |
| Tecnologías | La tecnología usada debe ser compatible con múltiples robots y de preferencia open-source que no suponga inconvenientes en procesos de investigación futuros o dependencia a un software en específico. |

Cada uno de los requerimientos posee una relevancia fundamental en el sistema, en específico, la gestión y monitoreo en tiempo real son requerimientos funcionales, mientras que los otros requerimientos se enfocaron en mejorar la experiencia del usuario, en el uso de la arquitectura y visualización de parámetros.

## 2.4 Soluciones

### 2.4.1 Soluciones alternativas

En base a los requerimientos del laboratorio y el análisis previo de la temática en el marco teórico descrito en el capítulo anterior, se consideraron varias soluciones alternativas que fueron evaluadas en base a las necesidades específicas y las herramientas disponibles. Entre las diferentes alternativas se encuentran:

- **Alternativa 1**: Software encargado de gestión de tareas y rutinas en un MRS mediante una arquitectura centralizada; en esta, los robots reciben las órdenes del operador.
- **Alternativa 2**: Software encargado de gestionar tareas y rutinas en un MRS mediante una arquitectura de comunicación descentralizada, que permite una comunicación bidireccional de los robots, basado en una previa configuración por parte del operador.

- **Alternativa 3**: Software open-source con funcionalidades de creación, edición y ejecución de rutinas y tareas de un MRS, además incluye un control de decisión automático en base a una previa configuración del operador.

### 2.4.2 Criterios de selección

Con el fin de determinar la solución adecuada, se establecieron criterios de selección que consideraron tanto las necesidades del laboratorio como las limitaciones técnicas y económicas. A continuación, se describen los criterios usados con su respectiva justificación:

- **Facilidad de integración con tecnologías actuales:** El sistema debe poseer la capacidad de adaptarse de forma sencilla a herramientas actuales, estables o en desarrollo, siendo ejemplo de ello, los algoritmos de visión por computadora o localización de robots ampliamente usados en los MRS.
- **Respuesta efectiva a fallos:** Este criterio evalúa directamente el funcionamiento de la arquitectura ante una eventualidad, que podría presentarse al realizar la ejecución de las tareas en cada uno de los robots, priorizando suspender únicamente la tarea que presenta fallas, es decir el robot no responde los tópicos o sufre perdida de suministro eléctrico, y no el resto de las tareas en ejecución.
- **Interfaz de usuario / visualización de datos:** El desarrollo de esta característica tiene como fin permitir que el usuario utilice de forma eficiente las funcionalidades de la arquitectura, resultándole sencillo e interactivo.
- **Tipo de robots aceptados:** Este criterio valora de forma directa la capacidad de la arquitectura de poder integrar al sistema diferentes tipos de control de UGV, tales como: diferenciales u omnidireccionales.

- **Fiabilidad en la ejecución de tareas:** Esta característica evalúa los mecanismos de acción que ejecuta la arquitectura ante un fallo, tanto en respuesta al usuario como en garantizar que las tareas y rutinas se cumplan en los tiempos previstos.
- **Costos de desarrollo y escalabilidad:** El factor económico en la implementación juega un papel fundamental en el desarrollo de la arquitectura, este criterio tiene en cuenta costos desarrollo del software y costos de dispositivos para generar la red inalámbrica necesaria para la comunicación con los robots y costos de mantenimiento que permitirán a la arquitectura mantenerse funcional y adoptar funcionales adicionales.

**Tabla 2.2**

*Criterios de selección*

| Criterio | Relevancia | Ponderación |
|---|---|---|
| Facilidad de integración con tecnologías actuales | 3 | 5 |
| Respuesta efectiva a fallos | 3 | 6 |
| Interfaz de usuario / visualización de datos | 1 | 3 |
| Tipo de robots aceptados | 2 | 4 |
| Fiabilidad en la ejecución de tareas | 3 | 6 |
| Costos de desarrollo y escalabilidad | 2 | 5 |

Las ponderaciones descritas en la Tabla 2.2. se evaluaron a partir de la incidencia del criterio en el funcionamiento de la arquitectura, formando 3 grupos:

- Facilidad de integración con tecnologías actuales, respuesta efectiva a fallos y la fiabilidad de la ejecución, determinan el funcionamiento óptimo de la arquitectura y su incumplimiento supone un riesgo para la seguridad de los equipos.

- El segundo grupo está conformado por el tipo de robots aceptados y los costos de desarrollo y escalabilidad, que se enfocan en brindar un sistema flexible y al usar tecnologías open-source el valor monetario de desarrollo se reduce.
- Finalmente, la interfaz de usuario brinda al cliente comodidad al usuario, pero no imposibilita el uso de las funciones de la arquitectura prescindir de este recurso de software lo cual fue la razón de su menor relevancia en el diseño.

### 2.4.3 Matriz de decisión

Los criterios permitieron enfocar las necesidades del usuario respecto a las limitaciones económicas y del entorno, por lo que en esta sección se contrasto con cada una de las soluciones alternativas propuestas, definiendo así la solución más optima bajo un procedimiento estructurado y preciso, como se muestra en la Tabla 2.3.

**Tabla 2.3**

*Criterios de selección Matriz de decisión entre las soluciones alternativas*

| Criterio | Ponderación | Solución alternativa 1 | Solución alternativa 2 | Solución alternativa 3 |
|---|---|---|---|---|
| Facilidad de integración con tecnologías actuales | 5 | 3 | 2 | 3 |
| Respuesta efectiva a fallos | 6 | 2 | 2 | 2 |
| Interfaz de usuario / visualización de datos | 3 | 1 | 1 | 2 |
| Tipo de robots aceptados | 4 | 3 | 3 | 3 |
| Fiabilidad en la ejecución de tareas | 6 | 2 | 1 | 3 |

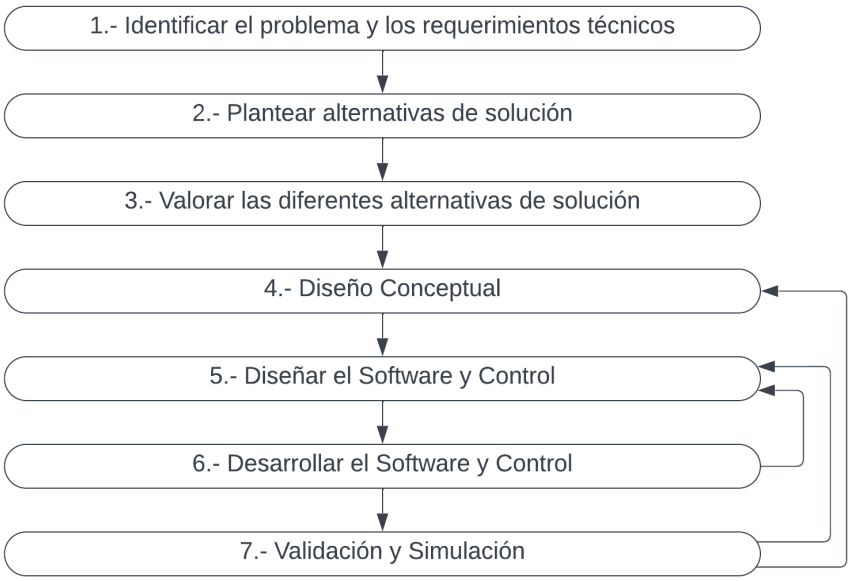
| | | | | |
|---|---|---|---|---|
| Costos de desarrollo y escalabilidad | 5 | 2 | 1 | 3 |
| Puntaje sin peso | | 12 | 10 | 16 |
| Puntaje con peso | | 64 | 48 | 78 |
| Prioridad | | 2 | 3 | 1 |

## 2.5 Etapas del diseño

El proceso de diseño presentado consta de diferentes etapas y es adecuado para el desarrollo de una arquitectura de software para el control maestro-esclavo de múltiples robots usando ROS2 Humble. En la Figura 2.1, se observan las diferentes etapas del diseño, la identificación del problema y sus requerimientos técnicos iniciales (etapa 1) son importantes para asegurar que las necesidades del sistema sean plenamente entendidas. La alternativa de solución (etapa 2) y su valoración (etapa 3) permiten explorar propuestas y elegir entre todas ellas la mejor según los criterios de selección. El diseño conceptual (etapa 4) muestra el sistema y su comportamiento principal. El diseño de software y control (etapa 5) y su desarrollo (etapa 6) son etapas finales y cruciales donde se definen y elaboran funcionales especializadas, aprovechando las mejores características del entorno de ROS2 Humble mediante la comunicación de sus nodos. Para finalizar, se tiene la validación y simulación (etapa 7) donde se asegura que el sistema cumpla con los requerimientos iniciales y funcione acorde a lo definido con el cliente, permitiendo iterar y mejorar el diseño en base a los resultados.

**Figura 2.1**

*Proceso de diseño para la elaboración de la solución*




## 2.6 Diseño conceptual

En el diseño conceptual de la arquitectura de software para la gestión de tareas aplicando metodología maestro-esclavo, se compone de varios módulos que se comunican entre ellos haciendo uso de ROS2, tal como se muestra en la Figura 2.2. ROS2 ofrece una comunicación por topics, services y actions, en algunos otros casos se necesitará comunicarse mediante información genérica como es el caso del namespace, lista de puntos de meta o la dirección del mapa que servirá para la navegación. La comunicación entre módulos es organizada para evitar enviar información innecesaria y con ello se eleve el procesamiento de los datos en el envío.

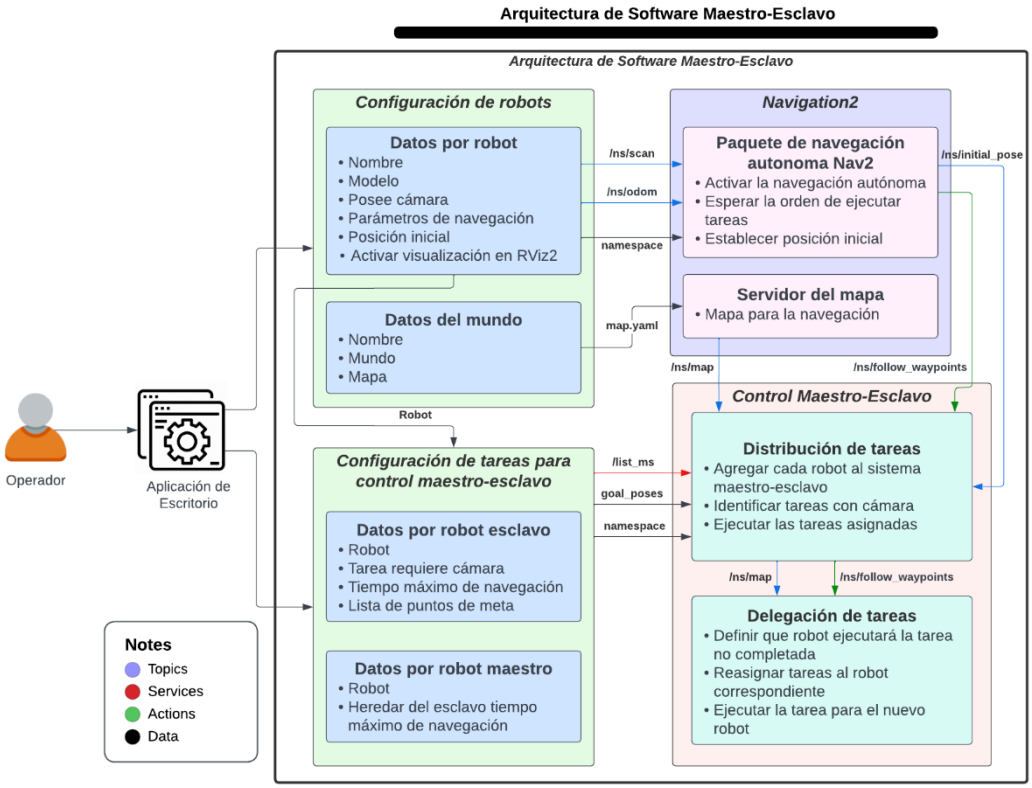
La arquitectura se encuentra desarrollada en su totalidad en el módulo “Control Maestro-Esclavo”, sin embargo, necesita de otros módulos como es el caso de la activación de la navegación autónoma de cada robot brindado por la librería interna de ROS2 llamada Navigation2, la cual permite construir un algoritmo de navegación desde el inicio o implementar algoritmos previamente desarrollados como el Algoritmo de Monte Carlo. Existen dos módulos de

configuraciones de los robots y configuraciones de tareas para el control maestro-esclavo, estos módulos sirven para saber que robots se usarán y designar los robots que cumplirán el rol de maestro o esclavo asignando las respectivas tareas que tendrán que realizar, también se puede definir un tiempo máximo de navegación.

El módulo principal cuenta con dos submódulos llamados “Control Maestro-Esclavo” y “Delegación de tareas”. El primero se encarga de armar la metodología maestro-esclavo con los robots previamente asignados, también identifica que tareas requieren el uso de cámaras para establecer como trabajos futuros de esta tarea solo a robots que contengan cámara, una vez que se tiene todo defino en el módulo se procede a ejecutar la navegación de cada robot de manera independiente. El segundo submódulo se basa en la delegación de tareas que no pudieron ser resueltas, es por ello que define el nuevo robot que ejecutara la tarea trasladando la tarea del anterior al nuevo robot, una vez que se tiene la tarea correctamente delegada se ejecuta la navegación del robot seleccionado.

**Figura 2.2**

*Arquitectura de Software para el control maestro-esclavo de las tareas*




## 2.7 Diseño de control

### 2.7.1 Patrón de diseño

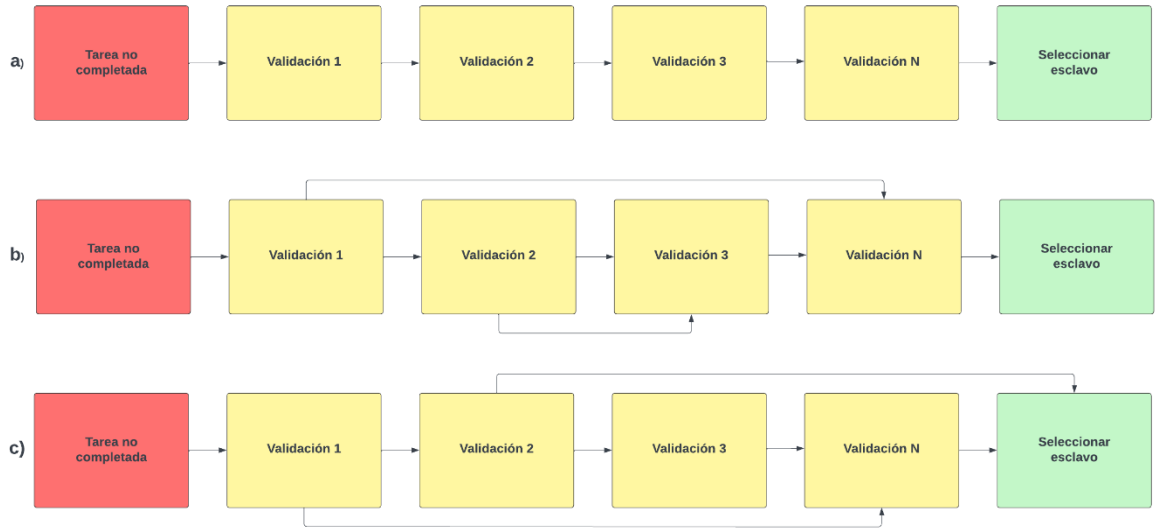
La arquitectura de software cuenta con un patrón de diseño llamado Cadena de Responsabilidades. Los patrones poseen la ventaja de elaborar arquitecturas de programación escalables y robustas en el tiempo, adquiriendo una fácil y rápida actualización del sistema si se quiere añadir o disminuir características.

El patrón de diseño Cadena de Responsabilidades permite elaborar una serie de validaciones para efectuar una tarea en específico, en la arquitectura de software desarrollada para delegar tareas que no fueron completadas se necesitan varias validaciones para conocer cuál es el mejor robot para efectuar la tarea. La cadena puede contener diferentes conexiones de validación, en la Figura 2.3, se observan diferentes formas de cómo se puede emplear el patrón de diseño. El

número de validaciones es ilimitado, por lo que si se desea implementar nuevas características al sistema no existirían problemas.

**Figura 2.3**

*Diferentes formas de aplicar Cadena de Responsabilidades. a) cadena directa, b) cadena con subdivisiones primer ejemplo, c) cadena con subdivisiones segundo ejemplo*




### 2.7.2 Delegación de tareas para robots esclavos

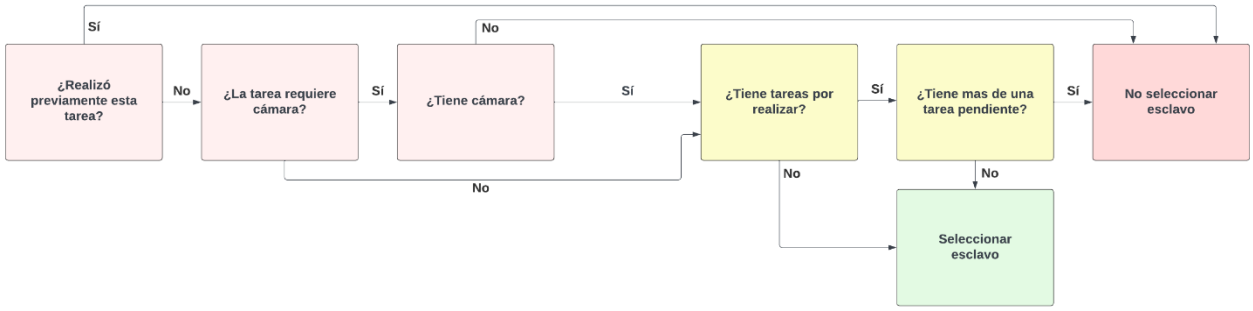
Cuando una tarea no se completa, se necesitan validaciones para conocer al mejor robot que podrá hacer la tarea. En la Figura 2.4, se muestra el orden de las validaciones, la arquitectura no permite que un mismo robot pueda repetir la tarea en más de una ocasión, para evitar que la tarea sea repetida un número ilimitado de veces lo cual podría causar que nunca termine de completar dicha tarea y la arquitectura se vuelva en un bucle infinito. Dentro de la arquitectura pueden existir tareas especializadas con cámara, por lo que en caso de que esto sea cierto se valida que el esclavo en cuestión cuente con cámara para proceder con las siguientes validaciones.

Una vez realizadas todas las validaciones previas, se procede con la validación de la navegación actual. El esclavo podrá ser seleccionado si no se encuentra realizando alguna otra tarea o si posee solamente una tarea por completar, de esta forma se evita que los esclavos estén

congestionados en su pila de tareas donde tal vez no se completen en un tiempo prudente y teniendo que esperar una gran cantidad de tiempo para completarlas.

**Figura 2.4**

*Delegación de tareas para robots esclavos*




### 2.7.3 Diagrama de flujo del control de la arquitectura

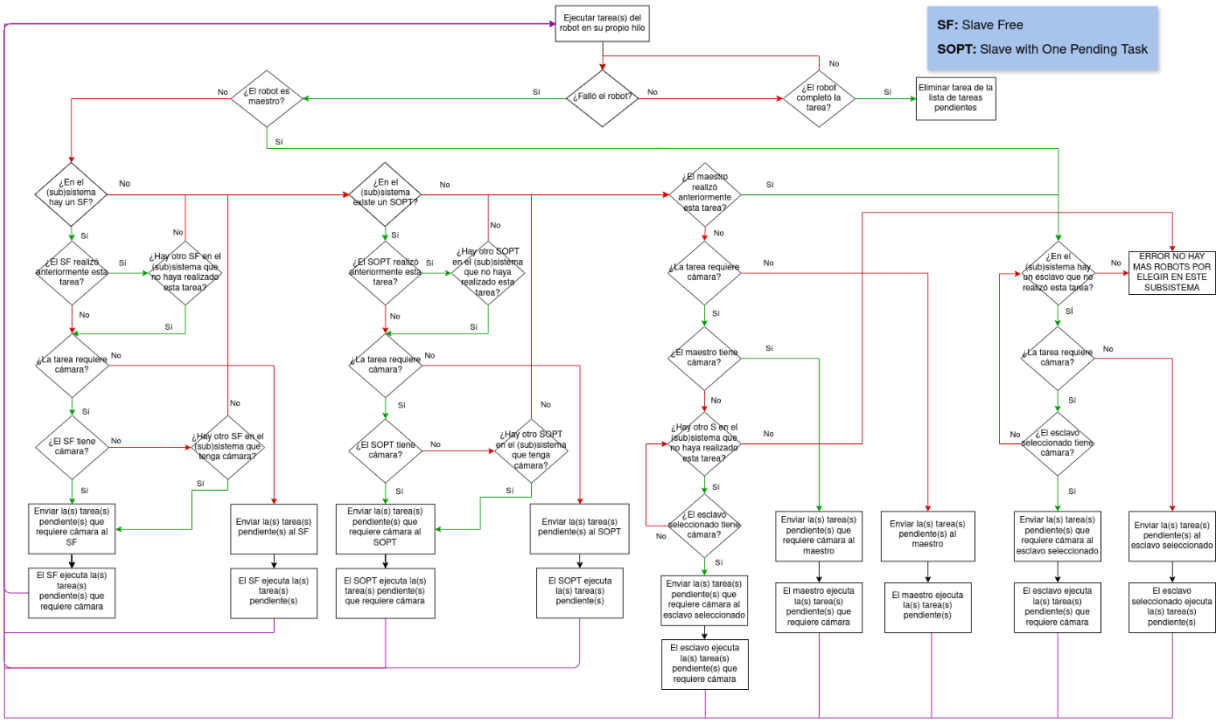
El control general de toda la arquitectura de software mantiene en todo momento las características para que sea escalable y robusto, no existen flujos discontinuos que provoquen cierres inesperados de la arquitectura al ejecutar una tarea en específica, tal como se muestra en la Figura 2.5. La arquitectura solamente posee dos salidas posibles, una de ellas es cuando la tarea es completada, lo que ocasiona que sea eliminada de la lista de tareas pendientes del robot, la otra salida es cuando ya no se posee algún otro robot dentro de la metodología maestro-esclavo, esto es un error natural que puede suceder debido a que los robots pueden dejar de funcionar en cualquier momento, por lo que no se tiene la certeza que siempre existirá un esclavo o maestro que pueda cumplir una tarea.

El flujo del control es individual por robot, ya sea esclavo o maestro, cada navegación tiene que correr en un hilo diferente, no se puede realizar una programación secuencial como normalmente sucede debido a que los robots tienen que funcionar y revisar tareas de manera individual en tiempo real. El maestro tiene la posibilidad de realizar tareas, sin embargo, solamente puede realizar tareas delegadas por sus esclavos, en caso de que el maestro no complete la tarea se

designa a un esclavo de emergencia sin importar su pila de tareas para que pueda ejecutar la tarea que no pudo ser completada.

**Figura 2.5**

*Diagrama de control para la asignación de tarea a un nuevo robot*




## 2.8 Diseño de aplicación de escritorio

El desarrollo de una aplicación de escritorio surgió bajo la premisa de satisfacer el criterio de una interfaz de usuario y la visualización de los datos del proceso durante la ejecución de las rutinas de los robots. El uso de ROS2 como base para el desarrollo de la arquitectura fue el factor decisivo para definir qué tipo de aplicación se desarrollaría, puesto que, la opción inicial era implementar una aplicación multiplataforma, sin embargo, dentro de las desventajas era el adaptar los drivers de comunicación para web, Android y iOS para ROS2.

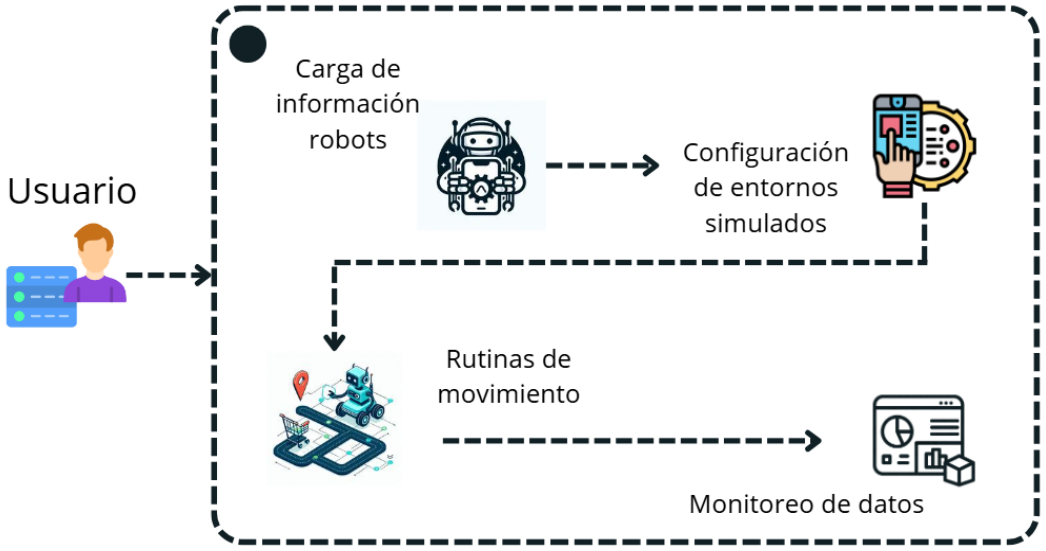
Por lo tanto, la solución fue desarrollar una aplicación de escritorio aprovechando que ROS2 se ejecuta en un sistema operativo como Ubuntu 22.04, ejecutando la aplicación de forma

nativa en el computador, siendo el único requisito que forme parte de la red inalámbrica donde se desplegara la arquitectura y los robots.

La aplicación se desarrolló en Python bajo la librería Flet, que permite el desarrollo de pantallas y widgets basados en Flutter, otorgándole un rendimiento óptimo a la par de pantallas visualmente atractivas. El usuario puede acceder a cada una de las funcionalidades de la arquitectura mediante un flujo de proceso, que se visualizan en la Figura 2.6.

**Figura 2.6**

*Flujo de proceso de la aplicación*




Cada módulo del proceso de la aplicación tiene su lógica de trasfondo, la cual se procederá a explicar de forma detallada:

- **Carga de información robots:** El usuario tiene la opción de guardar en un registro interno los modelos de sus robots e identificarlos con un nombre, siendo en este módulo donde el usuario ingresa la información de su robot necesaria para el despliegue como: nombre, modelo, rutas URDF y SDF (útiles para la ejecución en RViz y Gazebo); la utilidad de este módulo radica en poseer un respaldo de la información de los robots y su accesibilidad para configuraciones posteriores.

- **Configuración de entornos simulados:** En esta sección el usuario configurara las rutas de los mapas y mundos en donde la arquitectura desplegara cada uno de los robots, como en la sección anterior quedaran guardados en un registro local, además de permitir especificar datos como la posición en cada una de las coordenadas (x, y, z), su orientación (yaw) y opciones como la visualización en RViz por cada robot.
- **Rutinas de movimiento:** Una vez se han configurado correctamente los mundos el usuario tiene la alternativa de definir las tareas a realizar por cada robot y establecer los roles, en este caso sería maestro y esclavo por robot, para posteriormente ejecutar la rutina y poner en funcionamiento la arquitectura.
- **Monitoreo de datos:** Brinda una interfaz en la que el usuario podrá ver la imagen de cada una de las cámaras desplegadas por los robots, y adicionalmente, el estado de las tareas de cada uno de los robots, haciendo al usuario participe en la ejecución de tareas a modo de supervisor, cabe recalcar que esta funcionalidad estará activa cuando las rutinas de movimiento se encuentren ejecutándose**.**

## 2.9 Ecuaciones asociadas

Para describir el movimiento de los robots admitidos por el sistema, uno de los cuales son los robots diferenciales, es fundamental conocer como el movimiento de las llantas influye en la velocidad lineal, representando la relación entre la velocidad lineal de las ruedas izquierda y derecha, el radio de las ruedas r, y las velocidades angulares, como se observa en la ecuación 2.1.

$$\dot{x_{r1}} = \frac{1}{2} r \dot{\varphi_1} \qquad (2.1)$$

$$\dot{x_{r2}} = \frac{1}{2} r \dot{\varphi_2}$$

Por otro lado, el comportamiento cinemático global, descrita en la ecuación 2.2, necesario para el modelado de movimiento del robot en dos dimensiones, permite calcular su trayectoria en base a las velocidades de sus ruedas, requerimiento que toman en cuenta los fabricantes de robots al realizar el control e

implementación del robot en cuestión. (En esta sección no se centra en un robot en específico, ya que no es la finalidad de la arquitectura)

$$\dot{\xi}_I = R(\theta)\begin{pmatrix} \frac{r\dot{\varphi}_1}{2} + \frac{r\dot{\varphi}_2}{2} \\ 0 \\ \frac{r\dot{\varphi}_1}{2l} + \frac{-r\dot{\varphi}_2}{2l} \end{pmatrix} \qquad (2.2)$$

ROS2, mediante un marco de referencia, maneja un sistema de transformación denominado TF (Tree Frames), que describe la relación entre diferentes sistemas de coordenadas. Gracias a este sistema, el robot puede conocer su ubicación exacta en cada momento. La configuración del TF es realizada por el fabricante según las especificaciones del robot fabricado, la ecuación 2.3, ecuación 2.4 y ecuación 2.5 describen de forma simplificada como se sitúa en las coordenadas de un punto en específico (un sensor), respecto al sistema de referencia del robot:

$$H_i^C = \begin{pmatrix} R & P \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R_i^C & \begin{matrix} x_i \\ y_i \\ z_i \end{matrix} \\ 0 \quad 0 \quad 0 & 1 \end{pmatrix} \qquad (2.3)$$

$$H_C^0 = \begin{pmatrix} R_C^0 & \begin{matrix} x_c \\ y_c \\ z_c \end{matrix} \\ 0 \quad 0 \quad 0 & 1 \end{pmatrix} \qquad (2.4)$$

$$H_i^0 = H_C^0 \times H_i^C \qquad (2.5)$$

Por último, Nav2 (Navigation2) controla y planifica las trayectorias en las que el robot debe desplazarse de forma segura y eficiente, en las que se modelan ecuaciones en función de su rotación y traslación, junto a las aceleraciones máximas del robot, ya que se tienen en cuenta las capacidades físicas del robot, como se observa en la ecuación 2.6 y ecuación 2.7:

$$t_t \; tiempo\; de\; traslación$$

$$t_r \; tiempo\; de\; traslación$$

$$a_{t,\,max} = \max\frac{dv}{dt} \;\approx\; v_{max}/t_t \qquad (2.6)$$

$$a_{r,\,max} = \max\frac{d\omega}{dt} \;\approx\; \omega_{max}/t_r \qquad (2.7)$$

# Capítulo 3

## 3.1. Resultados y análisis

En esta sección se presentan los resultados de la arquitectura de software para la gestión de tareas de múltiples robots usando una metodología maestro-esclavo. En los ejemplos se presentarán casos con un sistema maestro-esclavo gestionando las tareas de robots diferenciales y omnidireccionales.

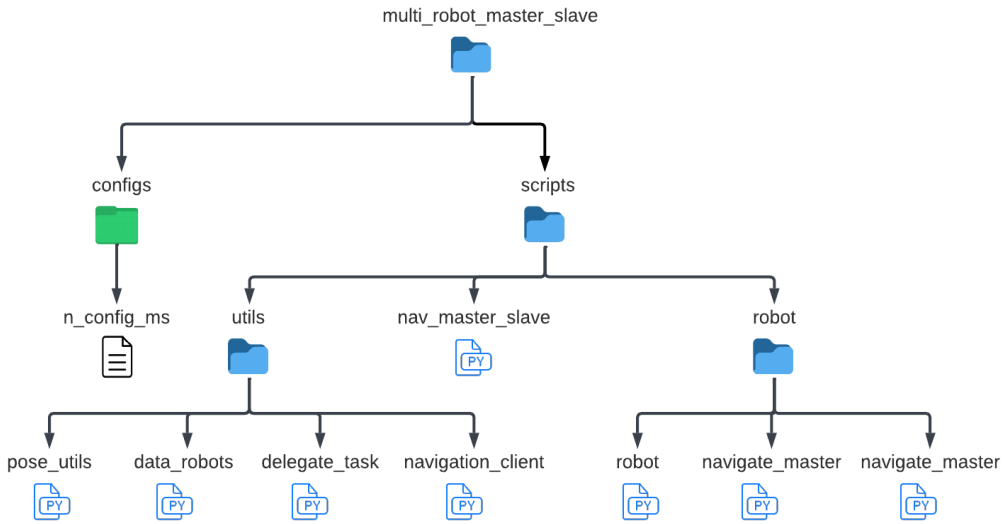
## 3.2. Arquitectura y Repositorio

### 3.2.1 Repositorio de la Arquitectura

El repositorio del proyecto se organizó de manera que facilitara el desarrollo y la gestión del sistema Maestro-Esclavo de múltiples robots, tal como se muestra en la Figura 3.1. En la raíz del directorio se ubicaron dos carpetas principales: "configs" y "scripts". En "configs", se incluyeron los archivos de configuraciones en estructura YAML, el cual contenía la información detallada sobre la distribución y configuración de los robots. La carpeta "scripts" se subdividió en dos subcarpetas: "utils" y "robot". En "utils", se almacenaron archivos de utilidades esenciales para mejorar la estructura de las tareas, como "navigation_client" y "delegate_task". En la subcarpeta "robot", se organizaron los scripts necesarios para la navegación tanto de los robots esclavos como de los maestros. El ejecutable principal "nav_master_slave.py", gestionaba la comunicación y control entre los robots maestros y esclavos.

**Figura 3.1**

*Estructura del repositorio del proyecto*

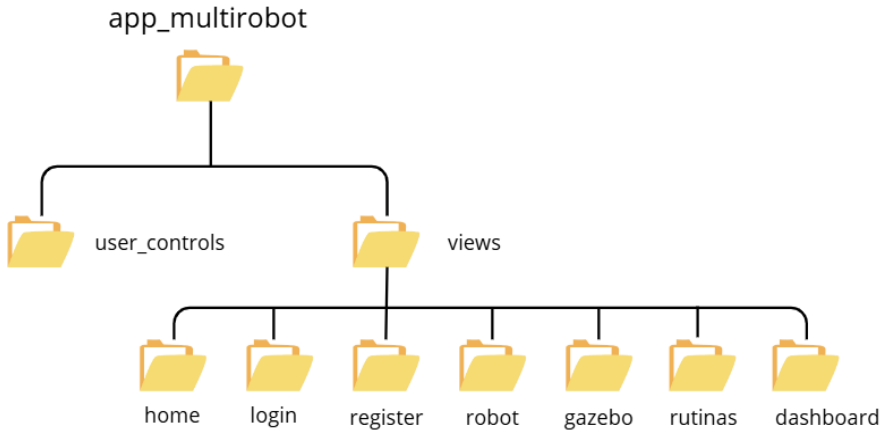



### 3.2.2 Repositorio de la aplicación de escritorio

La estructura de la aplicación de escritorio posee dos módulos principales: ‘user_controls’ y ‘views’, como se muestra en la Figura 3.2. El módulo ‘user_controls’ contiene cada una de las clases asociadas a la lógica del sistema, validaciones y comunicación con ROS2, necesarias para que la aplicación mantenga un desempeño optimo; mientras que, el módulo ‘views’, comprende cada una de las pantallas con las que interactúa el usuario, permitiendo realizar inicios de sesión, registro de usuario, registro de modelos y robots de forma personalizada, agregar en para el despliegue en simulación y configuración de rutinas con su respectivo monitoreo.

**Figura 3.2**

*Estructura del repositorio de la aplicación de escritorio*




## 3.3. Aplicación de escritorio

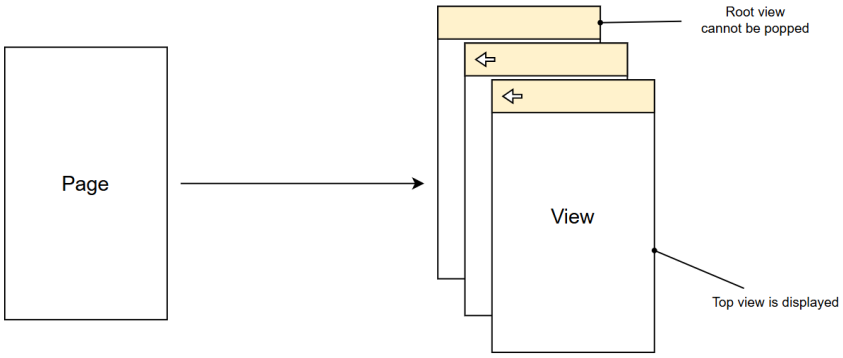
En las pruebas de concepto, el uso de la aplicación implicó el observar hasta qué punto se pueden configurar las funcionalidades de la arquitectura, lo que resultó beneficioso para el ciclo de codificación simultaneo de la aplicación y la arquitectura. La aplicación de escritorio se desarrolló para distribuciones Ubuntu, que es el sistema operativo sobre el cual se ejecuta ROS2. Esta decisión garantizó la compatibilidad de la arquitectura y la interfaz.

Mientras que, el lenguaje de programación elegido tanto para programar la parte grafica como la parte lógica de la aplicación fue Python en su versión 3.10, que brinda flexibilidad en la codificación, haciéndolo más fácil de implementar requerimientos funcionales de complejidad alta, y combinar la ejecución de ROS2 con su librería en Python llamado ‘rclpy’.

La parte grafica se implementó en base al módulo ‘flet’, que genera entornos gráficos basado en vistas, así cada pantalla con la que interactúa el usuario es una vista que se despliega sobre un entorno base llamado Page, como se muestra en la Figura 3.3.

**Figura 3.3**

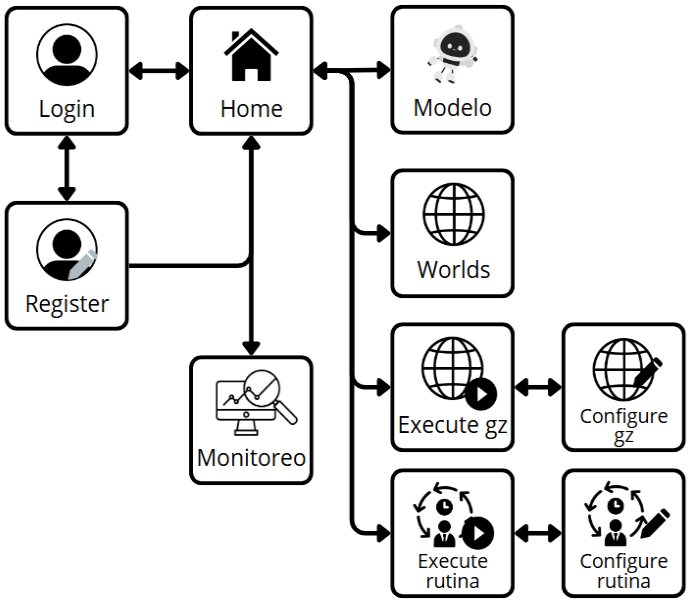
*Funcionamiento de views en Flet [21]*




La aplicación, en la Figura 3.4, posee 10 vistas que permitirán autenticar el acceso del usuario, registrar modelos y robots, creación de entornos simulados en gazebo, creación de rutinas y monitoreo de datos.

**Figura 3.4**

*Vistas de la aplicación de escritorio*

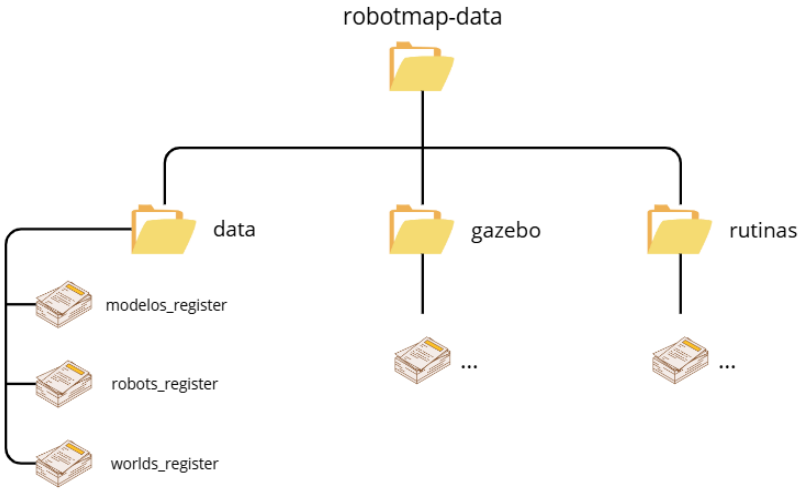



Mientras que, en la Figura 3.5, se muestran los directorios que alojan la información del usuario, modelos, robots, configuraciones de entornos y rutinas, gestionados de forma local y al ejecutar la aplicación por primera vez se construirá esta estructura en la carpeta raíz del sistema, los submódulos de robotmap-data se detallaran a continuación:

- **data:** Este directorio posee tres archivos YAML que corresponden al registro de los modelos, registros de los robots, e información de los mundos configurados por el usuario (imagen del mapa PGM y modelo SDF).
- **gazebo:** Almacena los archivos de parámetros de simulación del entorno a ejecutarse, es decir, mundo y robots, definiendo características como posiciones iniciales, uso de RViz.
- **rutinas:** Almacena las tareas que realizara cada robot en el entorno simulado, además de, configurar jerarquía, tiempos de ejecución de las tareas, y las posiciones objetivo.

**Figura 3.5**

*Directorio de datos de la aplicación*




## 3.4. Despliegue de la arquitectura

### 3.4.1 Importar robots

Para usar el sistema, primero se deben agregar robots en el registro general de la aplicación, proceso que se muestra en la Figura 3.6, en el capítulo anterior se abordó la relación entre robots y

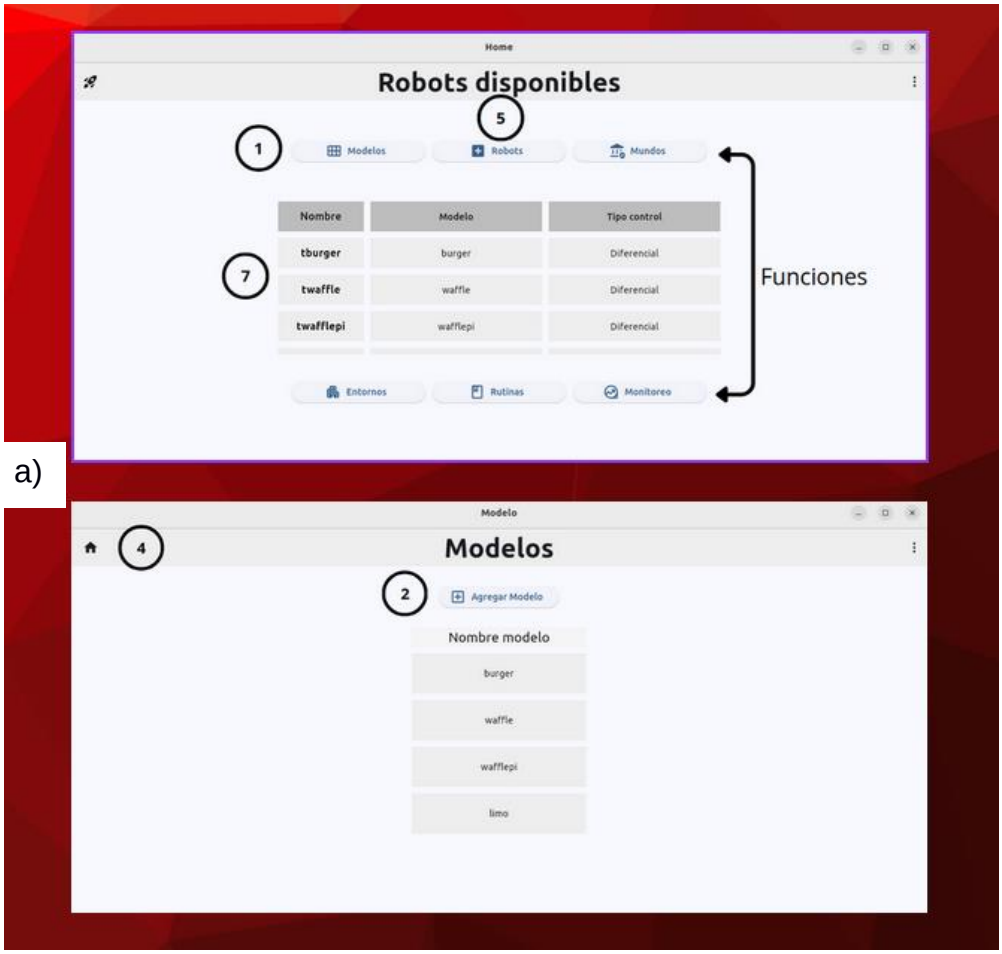
modelos, por lo que inicialmente se agrega un modelo, ejemplo de ello, pueden ser: Turtlebot Burger, Waffle o Waffle Pi, esto se visualiza en la sección 3.4.5 Entorno simulado.

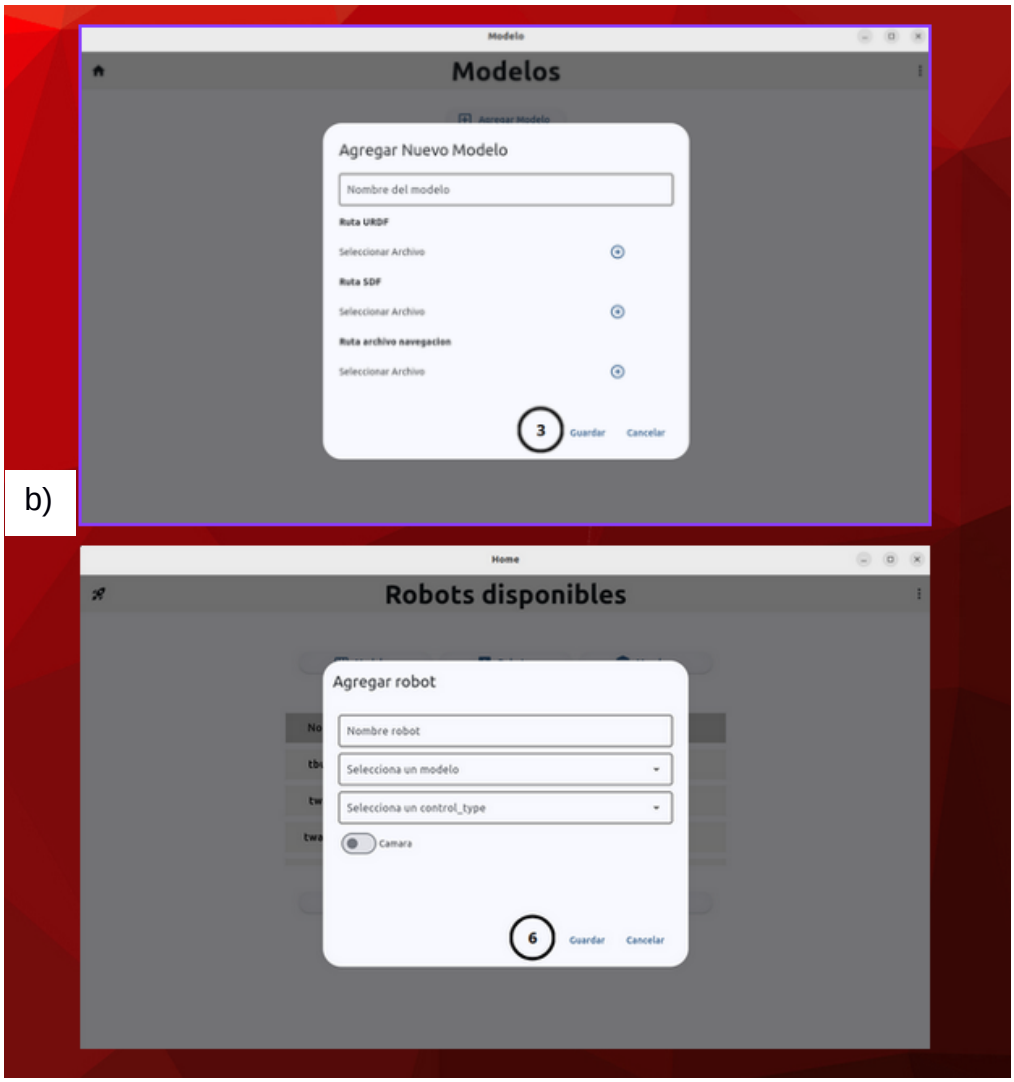
Luego, los modelos almacenados se mostrarán en una tabla con su respectivo nombre, siendo necesario agregarlo tal como se observa en la viñeta 2 en la sección a, y su respectiva configuración en la viñeta 3, donde se brindan datos como: nombre, rutas URDF, SDF y de parámetros de navegación.

Finalmente, se agrega el robot, visualizado en las viñetas 5 y 6, el cual recibe un nombre nuevamente, y los parámetros a configurar serán: nombre, modelo asociado, tipo de control y si posee cámara.

**Figura 3.6**

*Flujo de ejecución de importar robots. a) Pantalla de home y modelos. b) Cuadros desplegables de información de modelos y robots*



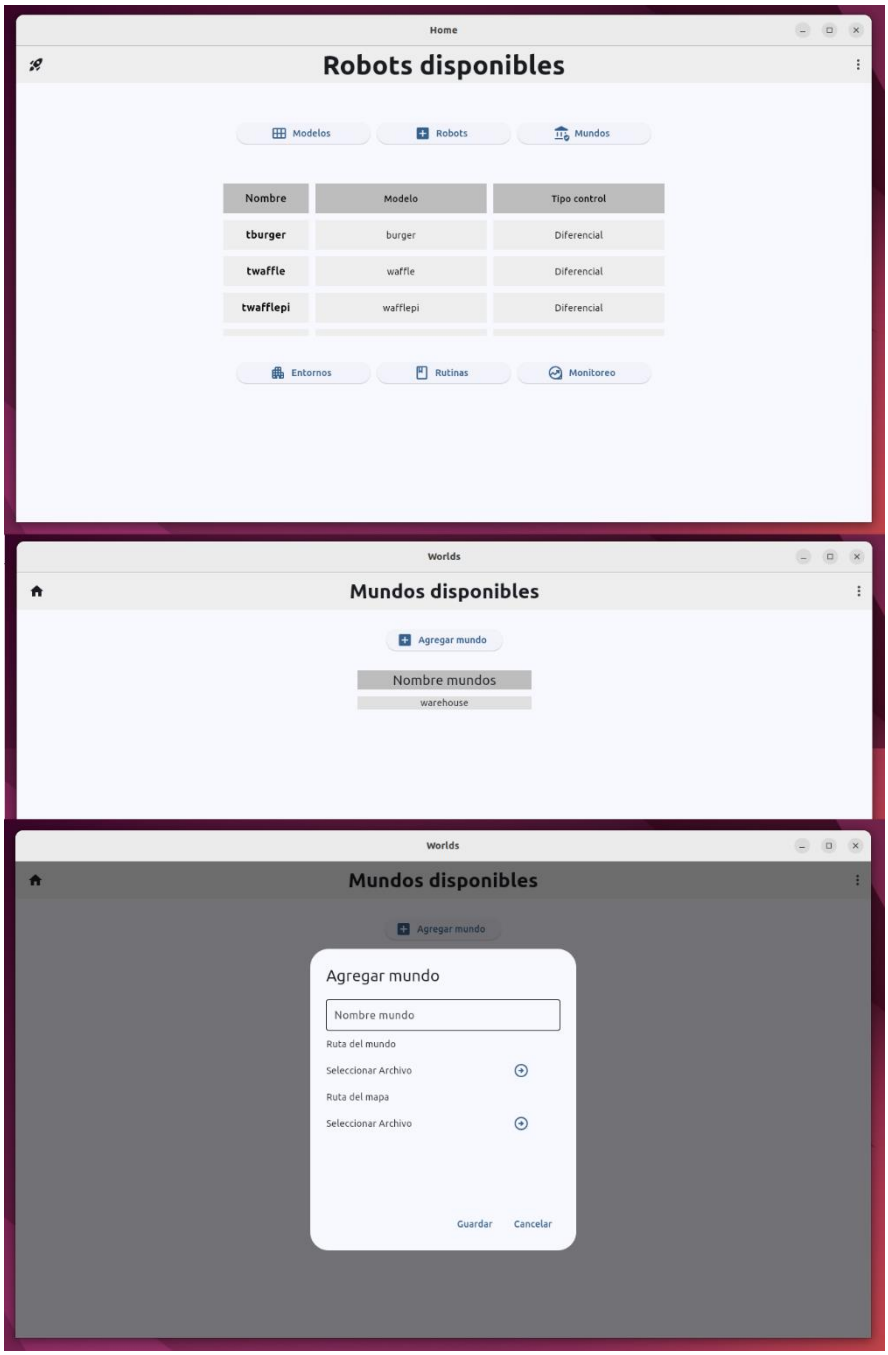



### 3.4.2 Configuración y despliegue de entornos

Este módulo consta de dos procesos: configurar el mundo, y configurar el entorno de simulación. La configuración del mundo parte desde la pantalla de inicio, tal como se describe en la Figura 3.7, al seleccionar la función de mundos el usuario observara una lista con los nombres de los mundos registrados, además de, poder agregar nuevos mundos dando información tal como: nombre del mundo, ruta del mundo y mapa generado para la navegación autónoma (son un archivo YAML y WORLD, respectivamente).

**Figura 3.7**

*Configuración de mundos en la aplicación*




Mientras que, los entornos se configuran siguiendo el flujo mostrado en la Figura 3.8, en la pantalla de entornos en Gazebo se listan las configuraciones previamente guardadas, con las opciones: play y stop, que iniciaran y finalizaran la ejecucion del entorno de simulacion.

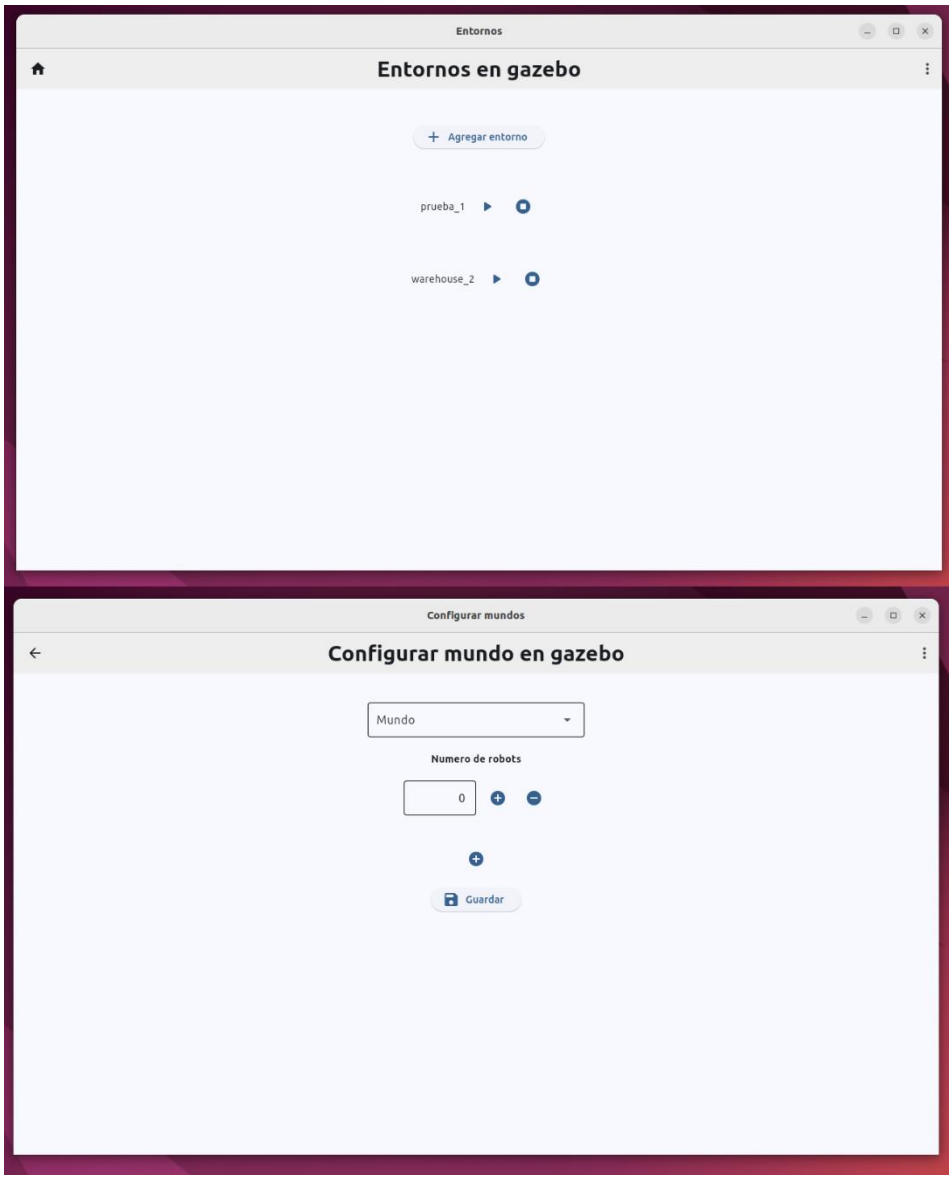
**Figura 3.8**

*Listado de entornos de simulación*


A continuacion, en la pantalla mostrada en la Figura 3.9, el usuario debe especificar parametros definidos para la configuracion de gazebo: el mundo, el numero de robots, la posición inicial de cada uno, junto con la configuracion de que si se ejecute RViz, siendo necesario la ejecucion de Gazebo.

**Figura 3.9**

*Configuración de entornos de simulación*



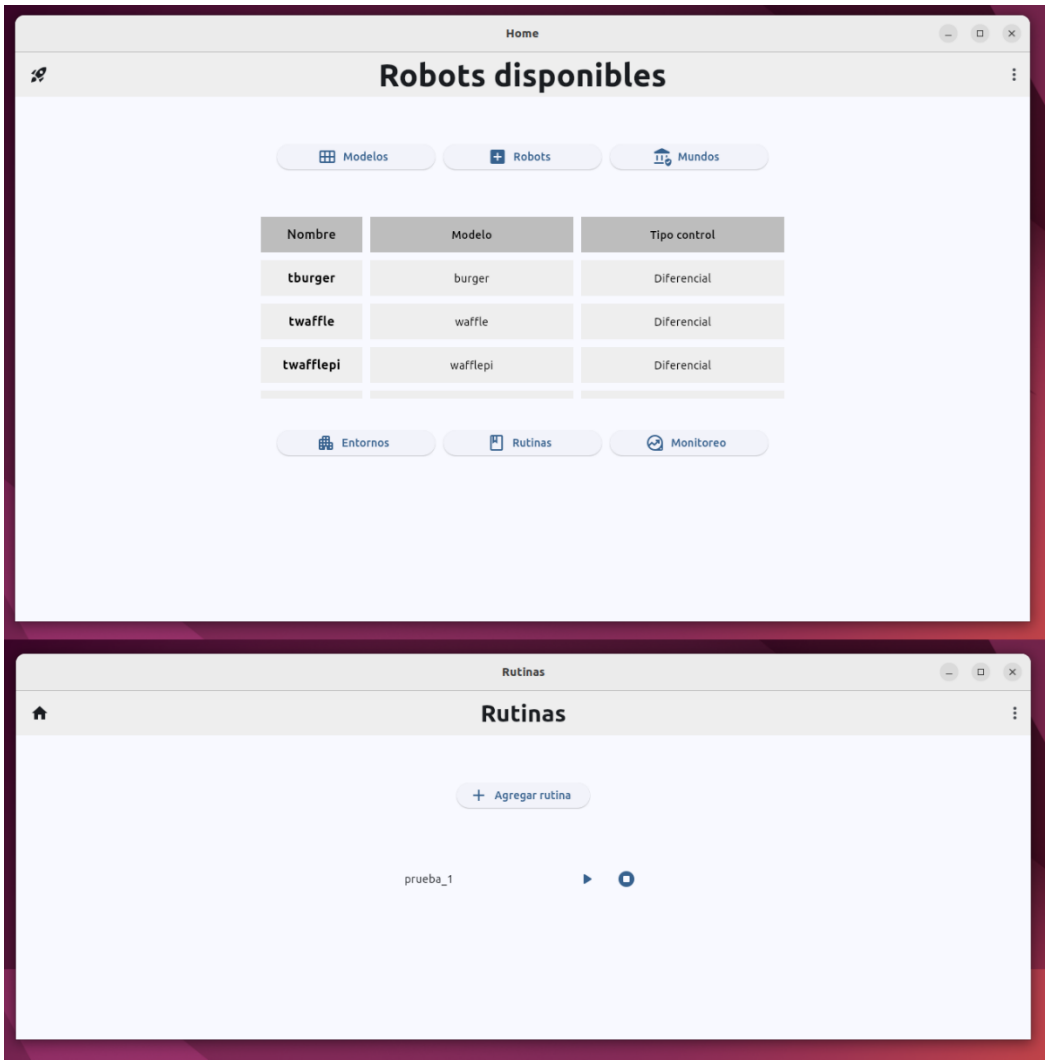

Finalmente, el despliegue de la simulación de Gazebo se realiza desde la pantalla de entornos y al momento de dar play en alguna de las opciones, se mostrará Gazebo con cada uno de los robots en sus posiciones configuradas, como se muestra a detalle en la Figura 3.10 corresponde a la sección posterior.

### 3.4.3 Configuración y despliegue de rutinas

La configuración de rutinas, como todas las funciones parten desde la pantalla de home, en la Figura 3.10 se visualiza un listado de rutinas preconfiguradas, en caso de haberlas, cada una de ellas tiene asociada un botón de inicio y detener; luego para agregar una nueva rutina, en la pantalla de “Rutinas” simplemente se tiene que seleccionar el botón “Agregar rutina” y se redireccionara a la siguiente pantalla de configuración.

**Figura 3.10**

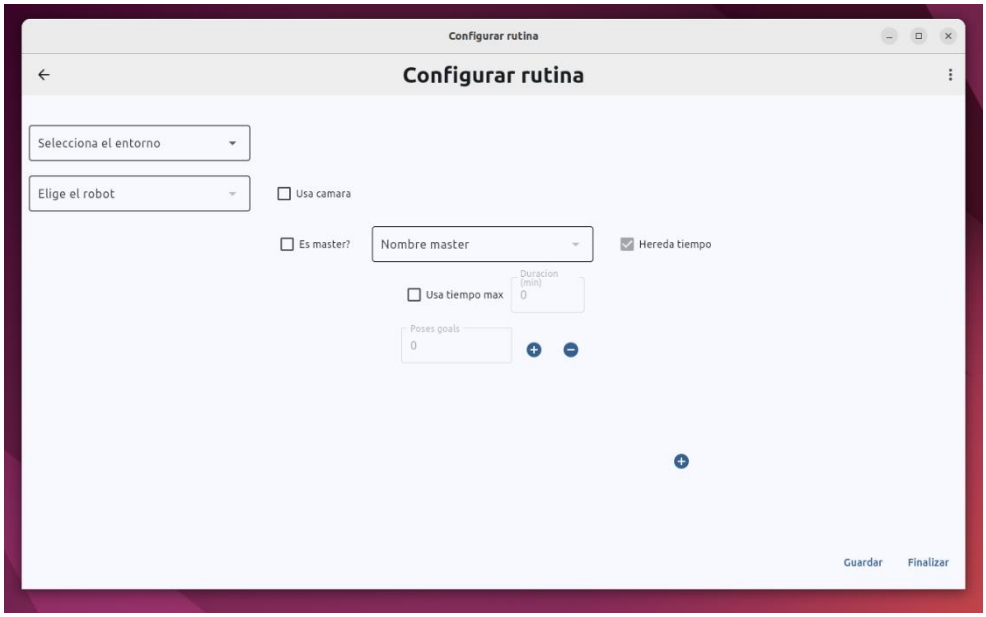
*Configuración de rutinas desde la aplicación*



A continuación, dicha pantalla de configuración de rutinas, Figura 3.11, en la cual el usuario puede configurar datos como: entorno de simulación, robot a configurar, uso de cámara, jerarquía del robot (si es maestro o esclavo), tiempos de ejecución de tareas y las posiciones a las que debe llegar en su tarea en cuestión.

**Figura 3.11**

*Ventana configuración de parámetros para las rutinas*



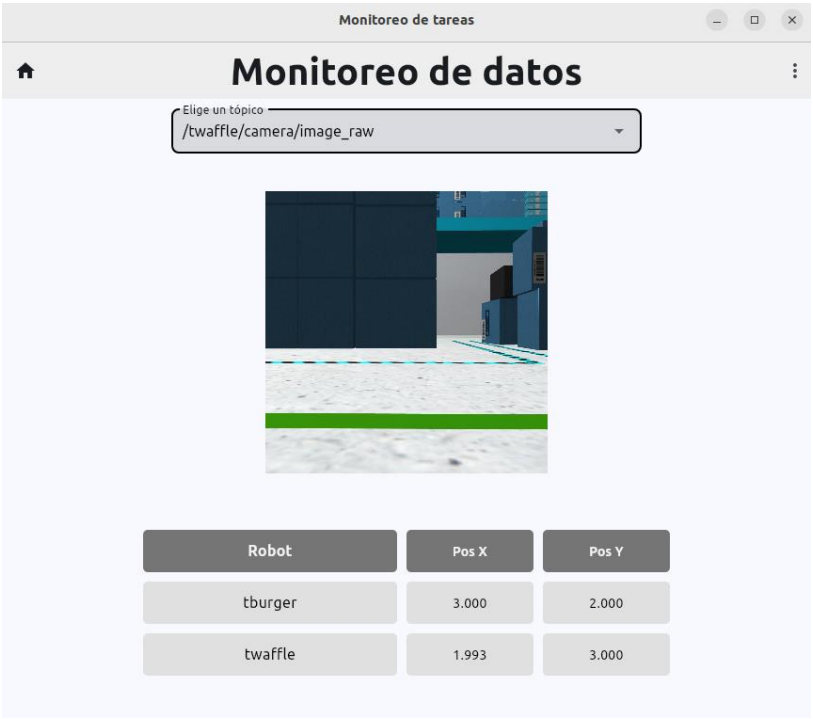

Por último, la ejecución de la rutina se realiza de la misma forma que en la ejecución de entorno, como se muestra en la Figura 3.12 de forma detallada en el mundo de gazebo previamente desplegado.

### 3.4.4 Monitoreo de robots

Este proceso tiene la finalidad de que el usuario tenga conocimiento sobre el estado de cada uno de los robots, la imagen de la cámara de los robots se mostraría en la pantalla y mediante selección el usuario puede visualizar por robot, mientras que, la posición del tiempo real y el avance de tareas completadas. En la Figura 3.12, se puede apreciar cómo se visualiza el monitoreo de datos en tiempo real de los diferentes UGVs.

**Figura 3.12**

*Ventana de monitoreo de datos*




### 3.4.5 Entorno simulado

En la Figura 3.13, se visualiza el entorno que se utilizó para probar la arquitectura de software usando la metodología maestro-esclavo para múltiples robots. El entorno fue realizado en Gazebo teniendo en cuenta parámetros físicos como la gravedad, fricción, colisiones, etc.

**Figura 3.13**

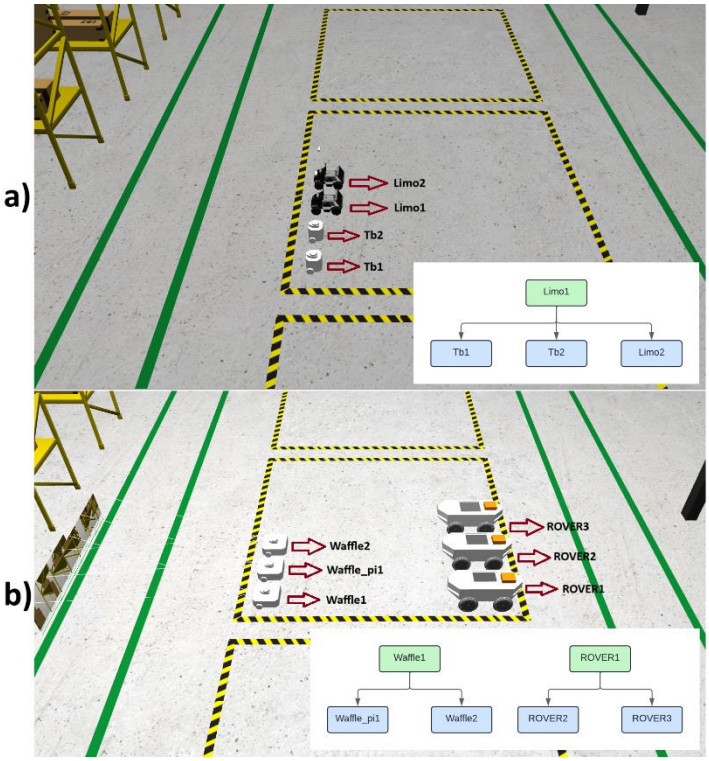
*Entorno warehouse para probar la arquitectura*

### 3.4.6 Distribución de sistemas maestro-esclavo

La arquitectura se elaboró para que sea capaz de ejecutar varios sistemas maestro-esclavo en un mismo entorno donde cada sistema puede poseer solo robots diferenciales, omnidireccionales o una mezcla entre ambos. En la Figura 3.14, se visualizan dos distribuciones diferentes de como estructurar nuestra arquitectura. En la primera forma se observa un solo sistema que contenía 2 robots diferenciales Burger y 2 omnidireccionales Limo donde el maestro era uno de los robots omnidireccionales, el cual se llamaba “Limo1”. La segunda forma contenía dos sistemas donde en ambos se apreciaban únicamente robots diferenciales donde un sistema contenía robots Waffle y Waffle Pi, el otro contenía únicamente robots ROVER, en esta distribución para el primer sistema el maestro era el Waffle1, mientras que para el segundo sistema era el ROVER1. La forma distribuir cada sistema es independiente, no hay una limitante para que los usuarios puedan seleccionar varios robots o diferentes modelos para un sistema en específico que se desee desarrollar.

**Figura 3.14**

*Distribuciones de la arquitectura para el control maestro-esclavo. a) arquitectura con un sistema maestro-esclavo, b) arquitectura con dos sistemas maestro-esclavo*




### 3.4.7 Ubicación de tareas

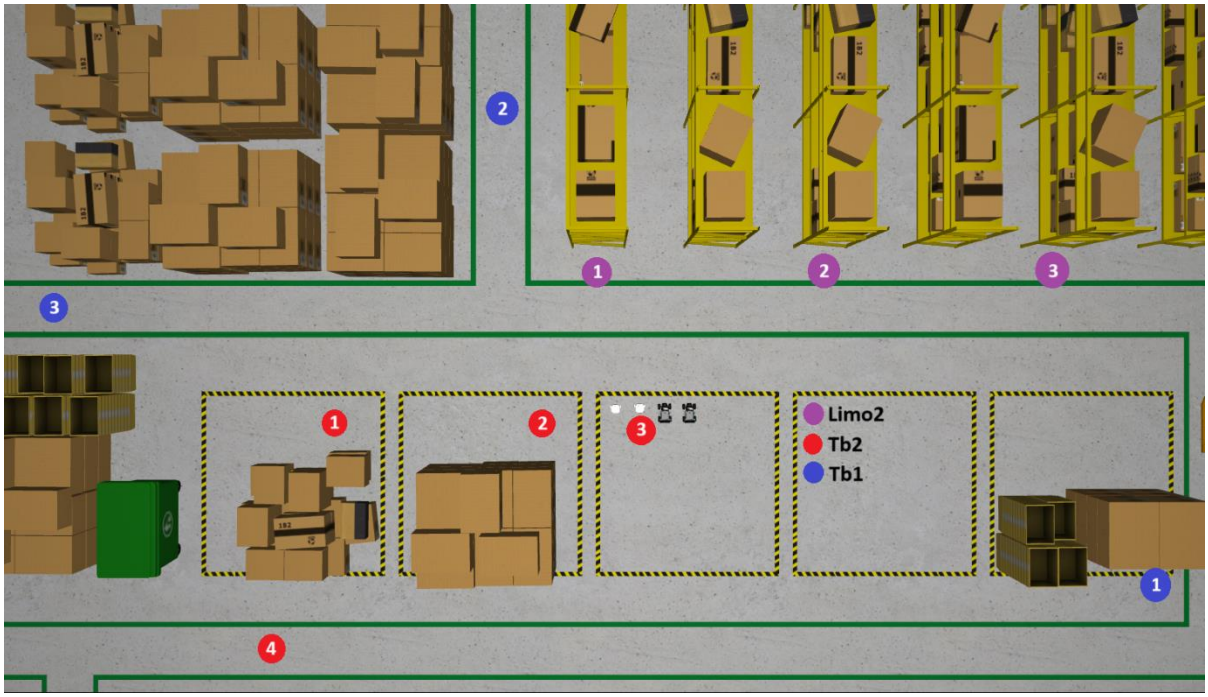
La arquitectura fue desarrollada para dejar totalmente personalizable el número de tareas, así como la ubicación de cada tarea que tiene que realizar el robot. En la Figura 3.15 se puede visualizar una configuración de tareas donde se estructuran únicamente para los esclavos debido a que se define al maestro solo para ejecutar tareas que no fueron completadas o para supervisión en caso de que se necesite, además, esta distribución maestro-esclavo es proveniente de la Figura 3.14a. Las ubicaciones de las tareas fueron establecidas en puntos donde el robot pueda llegar, en caso de seleccionar un punto donde se encuentre un objeto ningún robot podrá realizar la tarea porque en todo momento la navegación autónoma evita las colisiones.

Dependiendo de lo que sucedo en el transcurso de la navegación las tareas fueron completadas o delegadas a otros según las características necesarias. En secciones posteriores se

profundiza en más detalles de la arquitectura previo a su ejecución, en esta sección se define las ubicaciones de las tareas para cada robot.

**Figura 3.15**

*Listado de ubicación de las tareas para cada robot dentro de la arquitectura*




### 3.4.8 Organización de estructuras maestro-esclavo

Una vez que se establecido el entorno de ejecución, la distribución de los robots y asignación de la ubicación de las tareas, se procedió a realizar la configuración de las configuraciones para la navegación de cada robot esclavo o maestro dentro de la arquitectura. En la Figura 3.16, se detalla de manera visual como los robots fueron configurados, también se puede observar que todos los robots son altamente configurables.

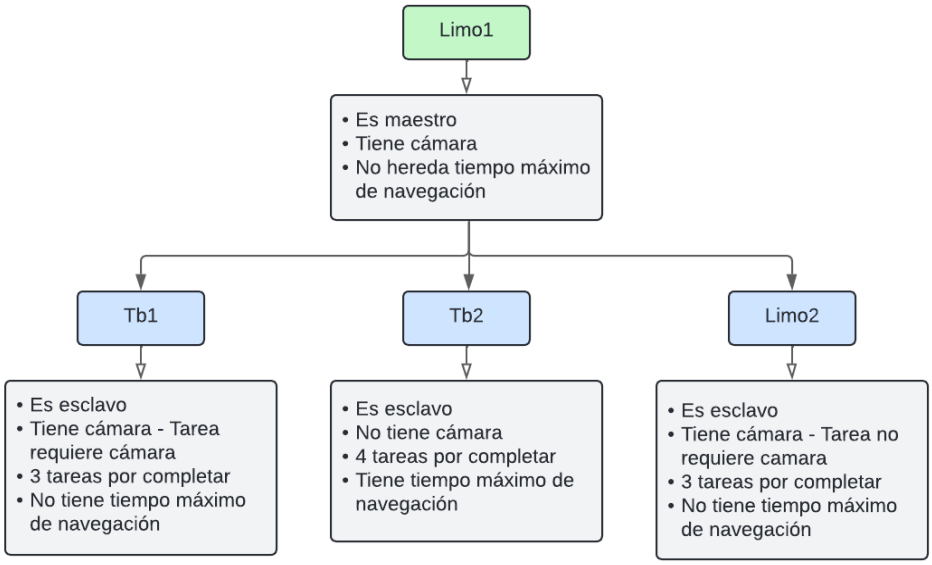
Al poseer una arquitectura de software uno de los beneficios es que todo es configurable para obtener los resultados deseados sin la necesidad de actualizar el código fuente. Además, al tener varias configuraciones se puede obtener diferentes resultados en la navegación del sistema,

el resultado presentado fue en base a una configuración donde se pudo observar mayores pruebas para la arquitectura para delegar tareas.

Había esclavos que tenían que desarrollar tareas especializadas con cámara como el Tb1, mientras que había otro esclavo llamado Limo2 que, aunque tenía cámara su tarea no requería de ella. El Tb2 tenía una configuración de un tiempo máximo de navegación para completar todas sus tareas. El maestro Limo1 posee cámara, por lo que en caso de que el Tb1 falle puede completar su tarea, también no hereda el tiempo máximo de navegación de sus esclavos por lo que si falla el Tb2 él se tomara el tiempo que crea necesario para terminar las tareas pendientes.

**Figura 3.16**

*Configuración de la navegación para los robots en la arquitectura de software*



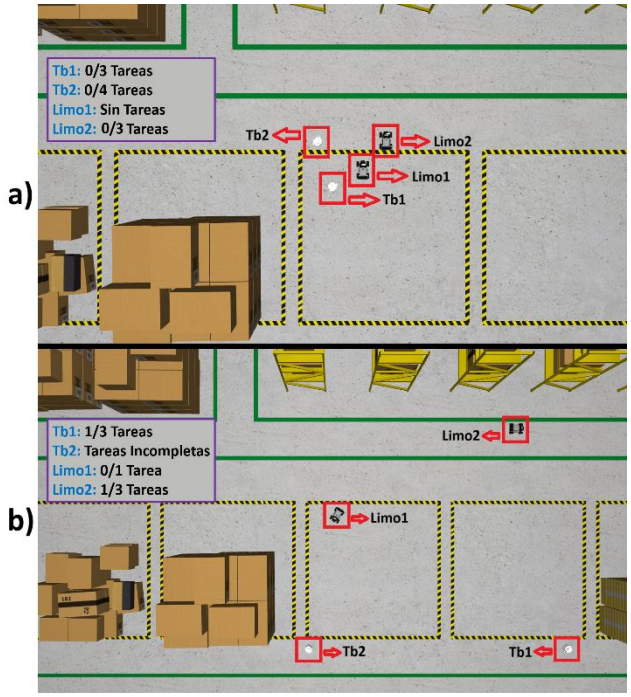

### 3.4.9 Despliegue de la arquitectura

Posterior a configurar toda la arquitectura con los robots deseados y asignando características de manera individual, continuando con la descripción de la Figura 3.16, se procedió al despliegue de la arquitectura para visualizar el funcionamiento en un warehouse.

Haciendo uso de la Figura 3.17a, se puede visualizar como los robots empiezan su funcionamiento. El maestro “Limo1” se queda en su lugar porque los maestros están configurados para no efectuar tareas iniciales, pero, está pendiente a cierta tarea delegada o para que el operador pueda teleoperar. En la Figura 3.17b, ocurre la primera entrada de la arquitectura porque “Tb2” estaba configurado con un tiempo máximo de navegación y dicho tiempo fue culminado, por lo que la tarea tuvo que ser delegada. La tarea no requiere cámara así que cualquier robot puede realizar la tarea, sin embargo, todos los esclavos están ocupados y tienen más de una tarea por completar, es por ello que la tarea es delegada hacia su maestro “Limo1”.

**Figura 3.17**

*Despliegue de la arquitectura. a) robots iniciando su funcionamiento, b) robot tb1 cumple con su tiempo máximo de navegación*



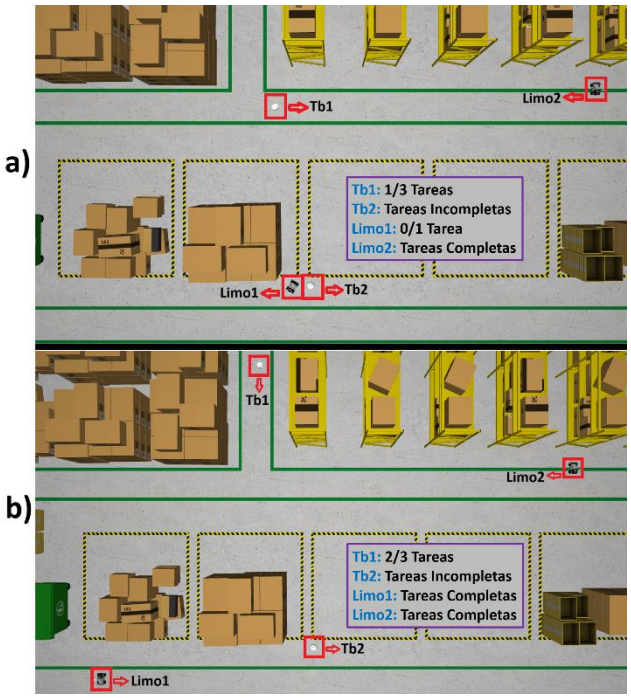

El esclavo “Tb2” como no pudo completar las tareas se queda en su última posición esperando una siguiente tarea delegada o accionada por el operador. El esclavo “Limo2” completa todas las tareas, tal como se observa en la Figura 3.18a, por lo que se queda esperando a que una

nueva tarea sea delegada y al tener cámara lo hace capaz de realizar cualquier tipo de tarea en la arquitectura. Hasta ese momento el esclavo “Tb1” solo ha completado una de sus tres tareas, mientras que el maestro “Limo1” aún no completa la tarea que fue delegada por “Tb2”. En la Figura 3.18b, se observa como el maestro “Limo1” termino de completar la tarea que fue delegada, dentro de los esclavos el único que se encuentra realizando tareas es “Tb1” al cual le falta únicamente una tarea por completar.

Todas las acciones que han sucedido hasta el momento son diferentes para cada arquitectura que se configure, si se selecciona una arquitectura con más de 1 sistema maestro-esclavo el funcionamiento será diferente, pero el centro de control seguirá siendo el mismo.

**Figura 3.18**

*Despliegue de la arquitectura. a) Limo2 completa todas las tareas, b) Limo1 completa la tarea delegada por Tb1*



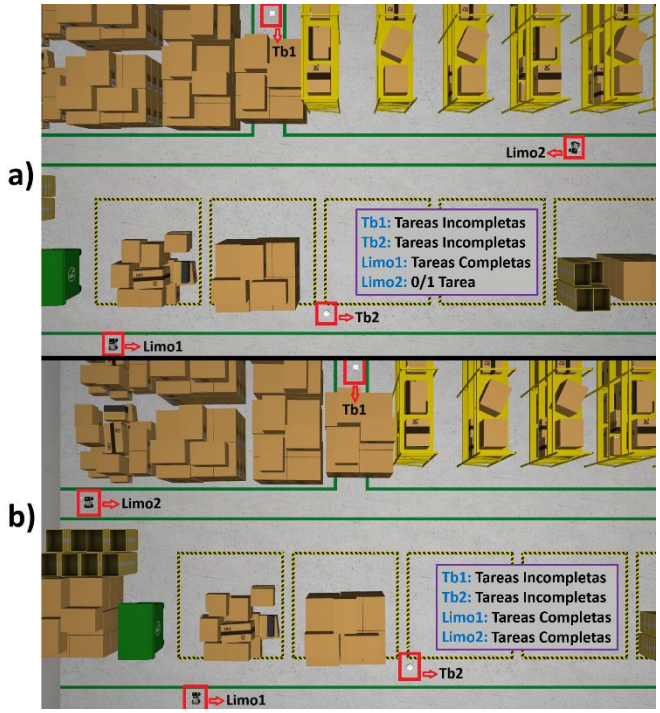

Para probar más acciones de control de la arquitectura se colocó un conjunto de cajas por delante del esclavo “Tb1” para que no encuentre una ruta de navegación y falle en la ejecución de sus tareas porque está encerrado. En la Figura 3.19a, se logra observar el suceso previamente

descrito. Debido a que la tarea del “Tb1” requiere cámara se necesita un esclavo o maestro que la contenga, es por ello que el control desechó al “Tb2” porque fue configurado como un robot sin cámara sin importar que se encontraba libre, así que se delegó la tarea al “Limo2” porque estaba libre y poseía una cámara para realizar la tarea.

Finalmente, en la Figura 3.19b, se observa como el esclavo “Limo2” completo la tarea que fue delegada dando por terminado todas las tareas de la arquitectura. En este punto se puede volver a ejecutar la misma arquitectura o seleccionar otra en caso que sea necesario con diferentes ubicaciones de tareas, maestros o esclavos, por lo que la metodología maestro-esclavo no se asigna con el encendido de los robots sino con el despliegue de la arquitectura.

**Figura 3.19**

*Despliegue de la arquitectura. a) Tb1 no encuentra una ruta de navegación, b) Todas las tareas asignadas a los robots son completadas*



## 3.5 Análisis de costos

La arquitectura de software desarrollada es código abierto, por lo que las industrias podrán usar y mejorar todo el sistema de acuerdo a las necesidades independientes de cada uno, sin embargo, para poder en marcha la arquitectura se necesita la adecuación y contratación de personal capacitado para llevar a cabo tareas de soporte, actualización y mantenimiento.

En la Tabla 3.1, se muestra la información la inversión inicial que se debe realizar para poder a prueba la arquitectura en un entorno real con robots reales, en esta tabla se omite el precio de los robots porque al ser una arquitectura soportada para todo tipo de UGV, entonces no se puede mencionar un precio establecido.

**Tabla 3.1**

*Inversión inicial para usar la arquitectura*

| Descripción | Cantidad | Precio Unit. (USD) | Precio Total (USD) |
|---|---|---|---|
| Programador de robots con ROS2 | 1 | $ 1000 | $ 1000 |
| Programador de aplicaciones | 1 | $ 1000 | $ 1000 |
| Computadoras | 3 | $ 900 | $ 2700 |
| Instalación de red para comunicación | 1 | $ 150 | $ 150 |
| Total | | | $ 4850 |

Una vez realizada la inversión inicial, se tendrá que realizar un pago mensual por diferentes actividades a realizar, ya sea variables o fijas. ROS y los UGV se comunican mediante una red, la cual normalmente es Wi-Fi, por lo que se debe de realizar un pago mensual fijo por el servicio de internet contratado, que normalmente en el mercado ecuatoriano un internet de buena calidad cuesta aproximadamente $50.

Dentro de los gastos variables, se tiene el pago de la planilla de luz y el mantenimiento o actualización de la arquitectura. En Ecuador, las industrias no tienen subsidio de luz, es por ello

que se coloca en la Tabla 3.2 un aproximado de lo que estaría gastando al mes todo lo asociado a la arquitectura, dependiendo directamente de la cantidad de robots que desea cargar, calidad de la red eléctrica, etc.

En el mantenimiento o actualización de la arquitectura algunas industrias suelen contratar proveedores que se encargan de realizar el trabajo en caso de que sea algo extenso, caso contrario las propias personas contratadas como programadores pueden realizar el trabajo. Algunas veces contratar proveedores de servicios para actualizaciones o mantenimientos como es el caso de RAMEL, asegura que el trabajo este correctamente realizado por su alta experiencia y un equipo robusto de trabajo tanto de programadores, electrónicos y mecánicos.

**Tabla 3.2**

*Costos fijos y variables mensuales*

| Descripción | Cantidad | Precio Unit. (USD) | Precio Total (USD) |
|---|---|---|---|
| Costo Fijo | | | $ 2050 |
| Servicio de Wi-Fi | 1 | $ 50 | $ 50 |
| Sueldo de programadores | 2 | $ 1000 | $ 2000 |
| Costo Variable | | | $ 1600 |
| Planilla de luz (promedio) | 1 | $ 100 | $ 100 |
| Actualización/Mantenimiento (promedio) | - | $ 1500 | $ 1500 |

# Capítulo 4

## 4.1 Conclusiones y recomendaciones

El desarrollo de una arquitectura para la gestión de tareas para sistemas de múltiples robots resulto satisfactorio al poder cubrir las necesidades de los MRS y ser una oportunidad de investigación en el área de control de tareas que se encuentra en un nicho importante junto a procesos como la localización de tareas, a continuación, se presentan las conclusiones y recomendaciones de este proyecto que involucra las diversas áreas que abarca la ingeniería mecatrónica.

### 4.1.1 Conclusiones

En el marco de los MRS, la gestión de las tareas de cada robot del sistema es crucial en la actividad general que realice el grupo maestro-esclavo, la arquitectura ofrece seguridad, control de las tareas, fiabilidad y alternativas durante la ejecución de la tarea del robot, la cual es indispensable en entornos industriales donde se necesitan que las tareas sean completadas y brinden un monitoreo constante de las actividades. Una vez finalizado el proyecto se procede a mencionar las conclusiones más destacadas:

- Se diseñó una metodología maestro-esclavo que puede ser usada en sistemas multirobots terrestres, la cual permite realizar, gestionar y delegar tareas. La arquitectura permite delegar tareas en base a una serie de configuraciones previamente establecidas para cierta tarea. Todo el proceso es automático, las personas únicamente tienen que seleccionar los robots, configurar el sistema maestro-esclavo y configurar las tareas que se van a ejecutar.
- Se logró desarrollar un sistema escalable a lo largo del tiempo para el control de los robots usando una metodología maestro-esclavo. La escalabilidad se asegura usando patrones de diseño como la cadena de responsabilidades, proporcionando de esa forma una vía para futuras implementaciones de acuerdo a las necesidades

de la industria, el uso de patrones de diseño permite elaborar arquitecturas empleando las mejores prácticas en el desarrollo de software para ingeniería.

- Se elaboró una arquitectura consumiendo la menor cantidad de recursos computacionales y distribuyendo de manera efectiva todas las tareas que se tienen que realizar. En la elaboración de la arquitectura se usaron librerías como Nav2 o rclpy que se encuentran establecidas en ROS y que tienen un gran soporte por la comunidad, se evitó emplear librerías relativamente nuevas o desconocidas donde su desempeño y funcionalidad aún no se prueba en entornos industriales.
- La interfaz gráfica de la arquitectura proporcionó una interacción agradable tanto en transiciones como en tiempos de respuesta en la ejecución de simulaciones y rutinas, donde se observó la mayor concentración de consumo de recursos computacionales. Además, se cumplió con las expectativas de brindar un monitoreo de los datos del MRS en funcionamiento, destacando la posición y las imágenes dando al operador una visión general de que pasa a cada momento e intervenir, en caso de ser necesario.
- En conclusión, la arquitectura de software desarrollada marca un nuevo campo de investigación en soluciones robóticas para uso industrial. Actualmente, los UGV realizan tareas de manera efectiva, sin embargo, la implementación de una metodología maestro-esclavo permite realizar, gestionar y delegar tareas con el fin que todas las tareas sean ejecutadas, además, se brinda una aplicación de escritorio para usar la arquitectura de una forma visual y brindando un monitoreo de los robots en ejecución. La arquitectura cuenta con buenas prácticas en el diseño de softwares lo que permite asegurar su escalabilidad y robustez.

### 4.1.2 Recomendaciones

El trabajo a futuro en la arquitectura de gestión brindara mejoras en funcionalidades en el campo de investigación robótica y de los MRS, siendo destacable remarcar las siguientes recomendaciones:

- La interfaz permitió que la arquitectura sea más sencilla de configurar, sin embargo, en términos de recursos computacionales, se puede economizar la ejecución de múltiples hilos que controlen los procesos por un servicio enfocado tipo API, usando un lenguaje enfocado en el ahorro de memoria como lo es C++ o Rust, que implican un desarrollo más complejo pero posible, en comparación a los plazos propuestos en la aplicación de escritorio realizada para esta investigación.
- Dentro de la acción de designar el procedimiento para realizar las tareas, la arquitectura podría integrarse a métodos que calculan el costo de realizar dicha tarea, así se lograría una gestión completa de las tareas, que es un campo de investigación en la robótica y MRS que se encuentra en fase de desarrollo.
- La configuración de los robots que serán parte de la metodología maestro-esclavo se realiza de forma manual por el operador, sin embargo, sería importante poder implementar algoritmos de Machine Learning para decidir automáticamente la configuración de la metodología.
- Elaborar un algoritmo para seleccionar puntos de trabajo en el mapa para la navegación autónoma usando visión por computadora, de esta forma las personas no tendrán que conocer la coordenada exacta del punto de la tarea, sino que mediante la propia aplicación podrán seleccionar donde se tendrá que movilizar el robot seleccionado para ejecutar la tarea.

# Referencias


[1] I. F. of Robotics, “World robotics 2023 report: Asia ahead of Europe and the Americas,” International Federation of Robotics, 2023. [Online]. Available: https://ifr.org/ifr-press-releases/news/world-robotics-2023-report-asia-ahead-of-europe-and-the-americas. [Accessed: Jul. 23, 2024].

[2] Instituto Ecuatoriano de Seguridad Social, "Normativa del IESS," pp. 271–272, 2016. [Online]. Available: https://sart.iess.gob.ec/DSGRT/norma%20interactiva/iess%20normativa.pdf.

[3] Allied Market Research, "Industrial Robotics Market Size, Share, Competitive Landscape and Trend Analysis Report by Type, by End User Industry, by Function: Global Opportunity Analysis and Industry Forecast, 2023-2032," Sept. 2023. [Online]. Available: https://www.alliedmarketresearch.com/industrial-robotics-market. [Accessed: Jun. 15, 2024].

[4] R. A. C. Schmidt, "Detección y diagnóstico de fallas en robots móviles cooperativos," 2004.

[5] S. Macenski, F. Martin, R. White, and J. Ginés Clavero, "The Marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[6] S. Nurmaini and B. Tutuko, "Intelligent robotics navigation system: Problems, methods, and algorithm," *International Journal of Electrical and Computer Engineering*, vol. 7, no. 6, p. 3711, 2017.

[7] Robotics Automation Mechatronics Engineering Laboratory (RAMEL). [Online]. Available: https://ramel.tech/. [Accessed: Sep. 2024].

[8] Escuela Superior Politécnica del Litoral (ESPOL). [Online]. Available: https://www.espol.edu.ec/es. [Accessed: Sep. 2024].

[9] Robot Operating System (ROS), "ROS," 2024. [Online]. Available: https://docs.ros.org/en/humble/Releases.html.

[10] "ROS, una palanca estratégica para el futuro de la robótica," Automation Review, Asociación Española de Robótica y Automatización, Jul. 2023. [Online]. Available: https://www.aer-automation.com/automation_review/ros-una-palanca-estrategica-para-el-futuro-de-la-robotica/. [Accessed: Sep. 2024].

[11] J. Kraaijenbrik, "Forbes," Forbes, 16 Jun. 2022. [Online]. Available: https://forbes.es/empresas/167359/que-es-la-industria-5-0-y-como-cambiara-las-empresas/. [Accessed: Jun. 19, 2024].

[12] R. R. Murphy, Introduction to AI Robotics. MIT Press, 2019.

[13] SMP Robotics, "SMP Robotics," 2024. [Online]. Available: https://smprobotics.com/technology_autonomous_mobile_robot/multi-agent-robotics-systems/. [Accessed: Sep. 2024].

[14] A. B. Roldán-Gómez, "Special Issue on Multi-Robot Systems: Challenges, Trends, and Applications," Applied Sciences, Madrid, 2021.

[15] Universitat Politècnica de Catalunya, "UPCommons," 2024. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2099.1/3330/34059-5.pdf?sequence=5. [Accessed: Sep. 2024].

[16] Real Academia de Ingenieria, «Diccionario Español de Ingenieria,» [En línea]. Available: https://diccionario.raing.es/es/lema/control-maestro-esclavo. [Accessed: Sep. 2024].

[17] Open-RMF Project. [Online]. Available: https://www.open-rmf.org/. [Accessed: Sep. 2024].

[18] OSRF, "Programming Multiple Robots with ROS 2," 2024. [Online]. Available: https://osrf.github.io/ros2multirobotbook/intro.html. [Accessed: Aug. 14, 2024].

[19] H. Khalil, S. Rahman, I. Ullah, I. Khan, A. Alghadhban, M. Al-Adhaileh, G. Ali y M. ElAffendi, "A UAV-Swarm-Communication Model Using a Machine-Learning Approach for Search-and-Rescue Applications," Drones, vol. 6, 2022.

[20] P. A. Z. Zhang, L. Q. Liu, and Y. Y. Zhang, "See Farther and More: A Master-Slave UAVs Based Synthetic Optical Aperture Imaging System with Wide and Dynamic Baseline," Optics Express, 2024.

[21] Flet, "Flet Docs," 2024. [Online]. Available: https://flet.dev/docs/getting-started/navigation-and-routing. [Accessed: Aug. 26, 2024].

**Apéndice A**

README del repositorio del MRS

# Multi Robot System

Multi Robot System developed for ROS2 Humble, allows the easy incorporation of several robot models with little configuration. There is no limit on the number of robots, if you incorporate UGV you can provide your parameter file for autonomous navigation.

## Features

- Unlimited number of robots
- Allows all types of robots
- Incorporation of robot navigation files
- Simple and fast configuration
- Supports robots designed in URDF, SDF or both

## Installation

Create a workspace

```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws/src
```

Clone this repository

```
git clone https://github.com/iesusdavila/multi_robot.git
```

Build the workspace with colcon

```
cd ~/ros2_ws
colcon build
```

## Configuration

### Input parameters for the world

For the world we have to enter the following parameters:

- Name of the world
- Route where the world is located
- World map (for autonomous navigation)

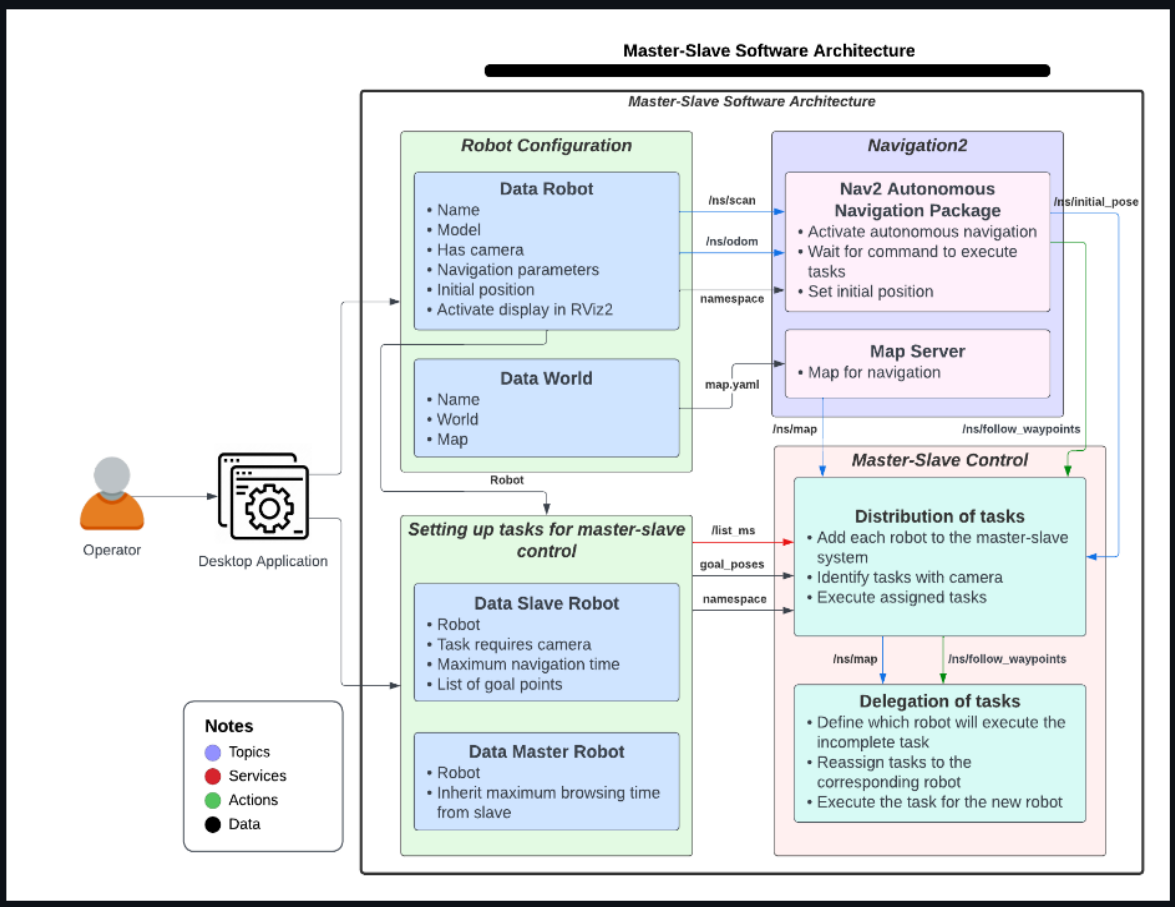
**Apéndice B**

README del repositorio de arquitectura maestro-esclavo

# Software Architecture Master-Slave

The development included the use of UGVs and ROS2 Humble, applying object-oriented programming techniques and robust design patterns. The results confirmed that, under the master-slave structure, the system completes all assigned tasks, even in the event of failure of one or more robots. In addition, an intuitive desktop application was designed, which facilitates the operation of the system without the need for specialized knowledge in ROS2. The proposed architecture is shown to be effective for managing and delegating tasks in terrestrial multirobot systems, establishing a solid foundation for future research.

## Software Architecture




## Installation

Create a workspace

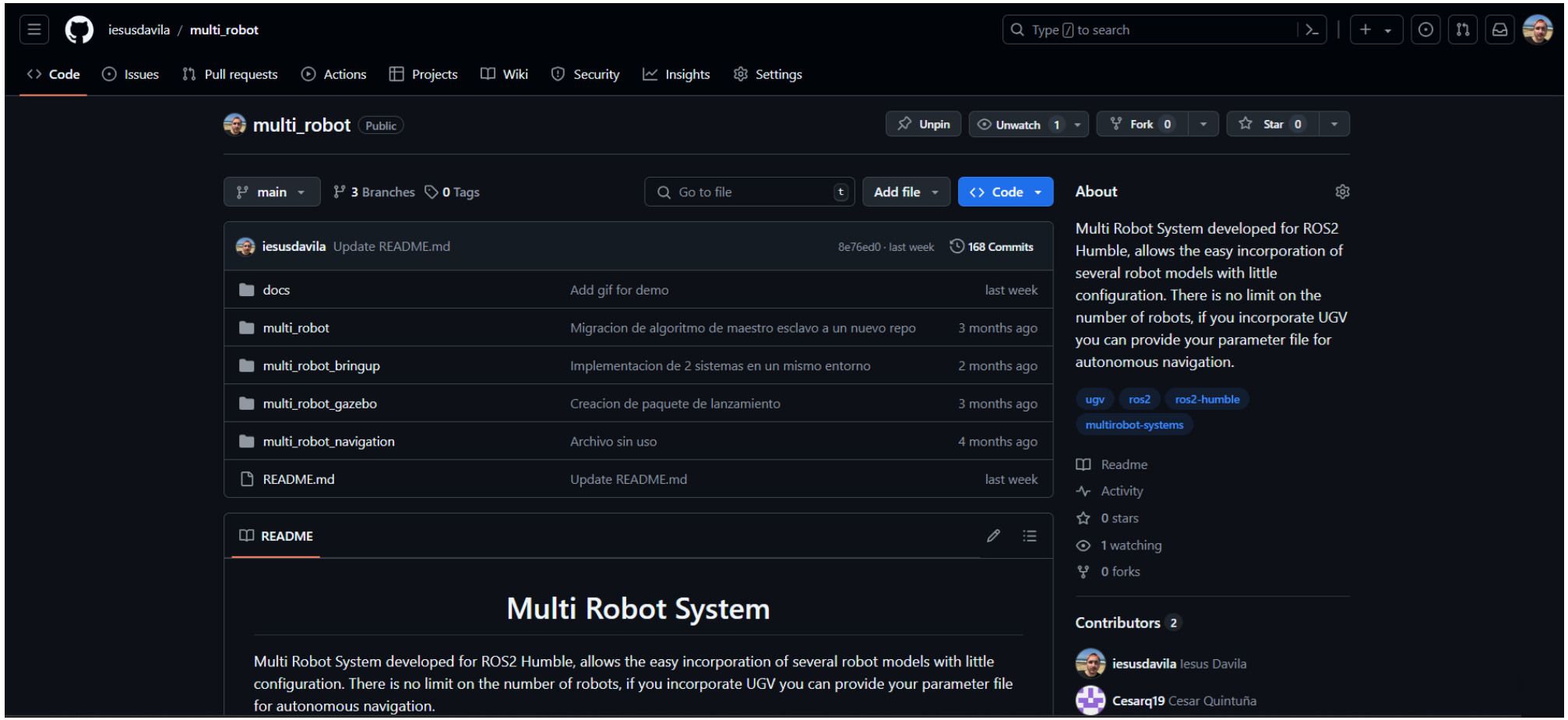
```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws/src
```

Clone this repository

```
git clone https://github.com/iesusdavila/master_slave_mrs.git
```

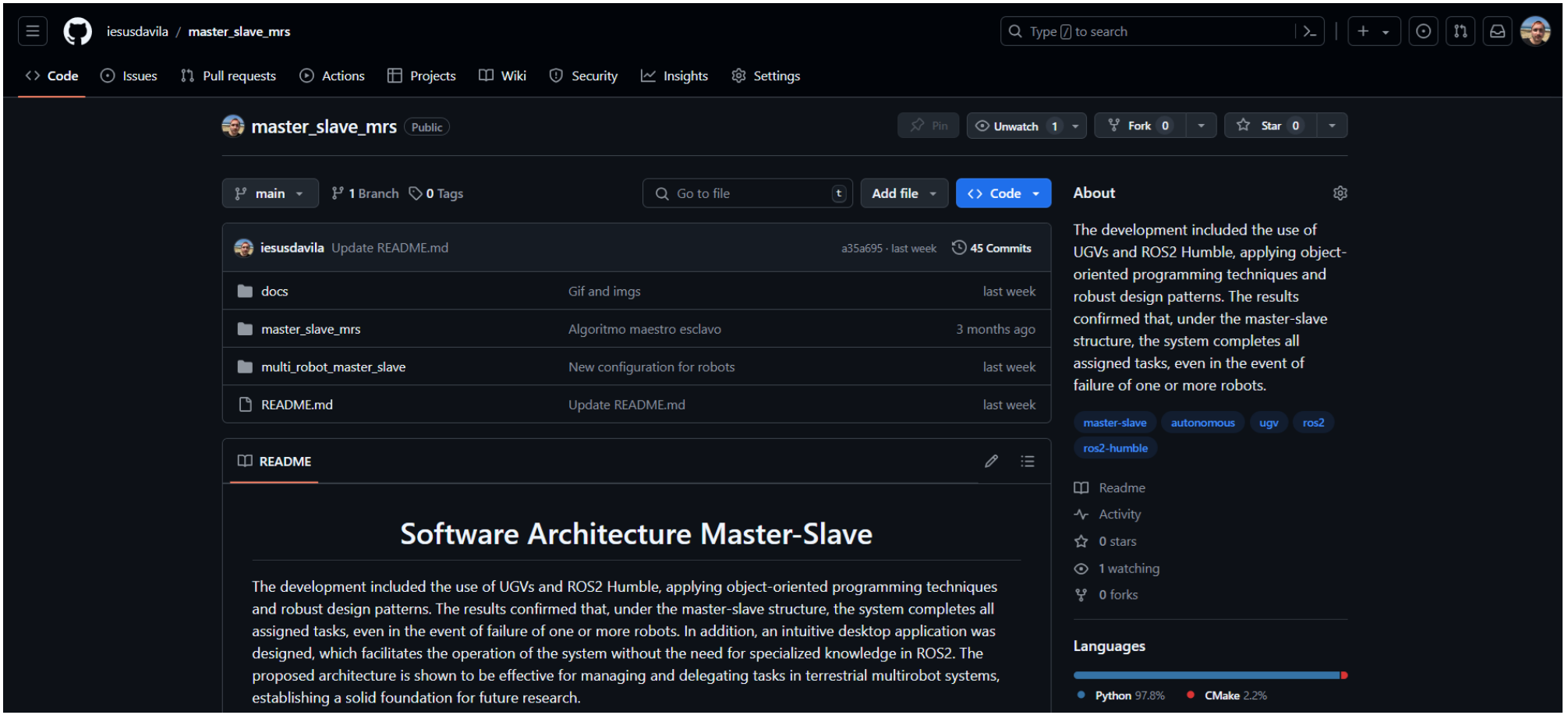
# Apéndice C

Repositorio del MRS



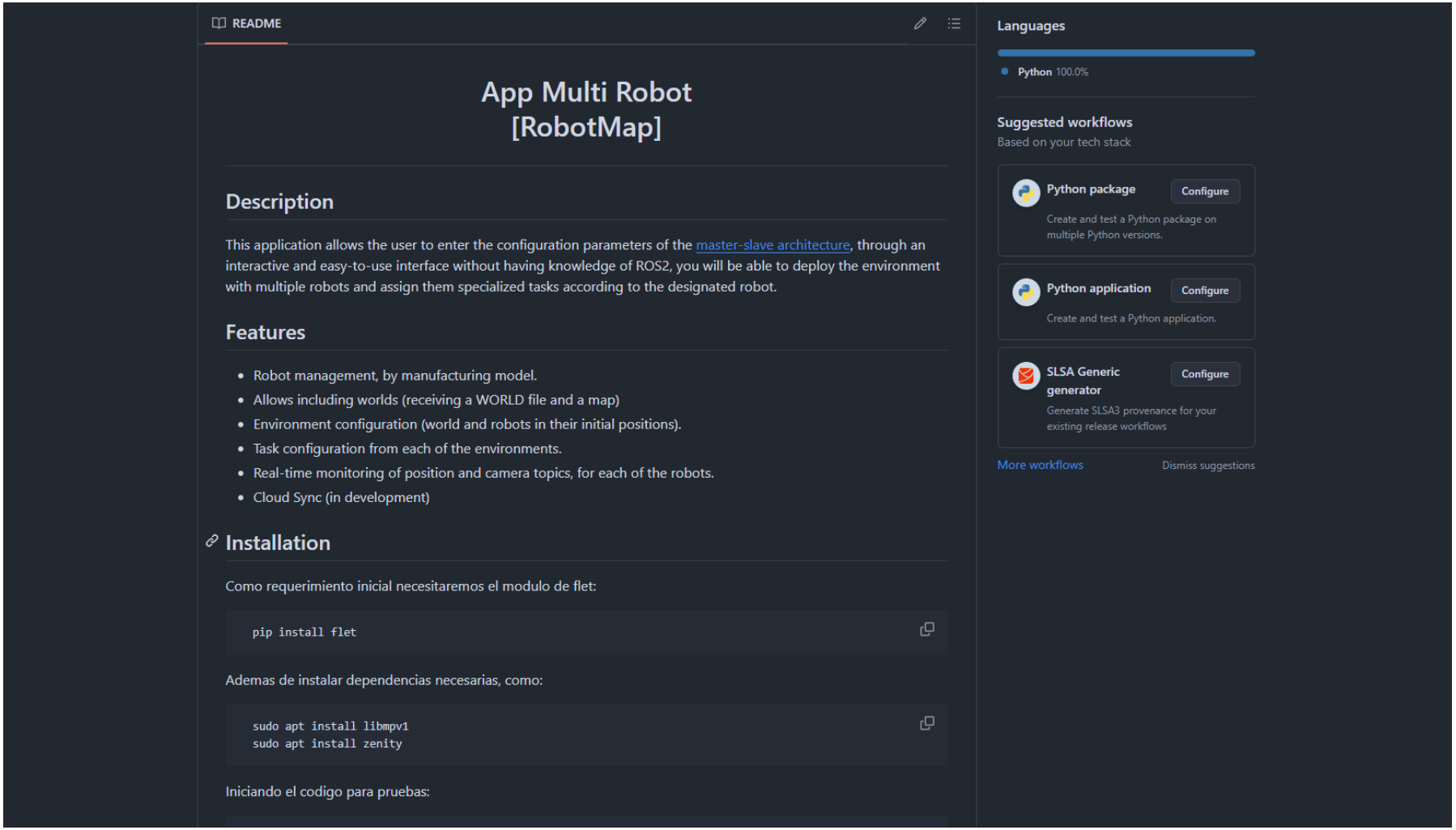
# Apéndice D

Repositorio de arquitectura maestro-esclavo



## Apéndice E

README de aplicación de escritorio



README

# App Multi Robot [RobotMap]

## Description

This application allows the user to enter the configuration parameters of the master-slave architecture, through an interactive and easy-to-use interface without having knowledge of ROS2, you will be able to deploy the environment with multiple robots and assign them specialized tasks according to the designated robot.

## Features

- Robot management, by manufacturing model.
- Allows including worlds (receiving a WORLD file and a map)
- Environment configuration (world and robots in their initial positions).
- Task configuration from each of the environments.
- Real-time monitoring of position and camera topics, for each of the robots.
- Cloud Sync (in development)

## Installation

Como requerimiento inicial necesitaremos el modulo de flet:

```
pip install flet
```

Ademas de instalar dependencias necesarias, como:

```
sudo apt install libmpv1
sudo apt install zenity
```

Iniciando el codigo para pruebas:

Languages

Python 100.0%

Suggested workflows

Based on your tech stack

Python package — Configure

Create and test a Python package on multiple Python versions.

Python application — Configure

Create and test a Python application.

SLSA Generic generator — Configure

Generate SLSA3 provenance for your existing release workflows

More workflows  Dismiss suggestions

## Apéndice F

Repositorio de aplicación de escritorio