

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD DE INGENIERÍA
PROGRAMA DE DOCTORADO EN INGENIERÍA

Énfasis en ciencia de la información y el conocimiento

MODELO DESCENTRALIZADO DE NAVEGACIÓN PARA SISTEMAS ROBÓTICOS MULTI-AGENTE COOPERATIVOS

Andrés Camilo Jiménez Alvarez

Tesis de grado para optar por el título de Doctor en Ingeniería

Director
Ph.D. Sandro Javier Bolaños

Codirector
Ph.D. Vicente García Díaz

Bogotá, Colombia, Noviembre 20 de 2018

COMISIÓN DE DOCTORADO

Esta tesis, titulada "MODELO DESCENTRALIZADO DE NAVEGACIÓN PARA SISTEMAS ROBÓTICOS MULTI-AGENTE COOPERATIVOS", escrita por Andrés Camilo Jiménez Alvarez, ha sido aprobada en cuanto a estilo y contenido intelectual.

Hemos leído esta tesis y la aprobamos,

Ph.D.
Jurado 1

Ph.D.
Jurado 2

Ph.D.
Jurado 3

Ph.D.
Sandro Javier Bolaños
Director

Andrés Camilo Jiménez
Estudiante

Agradecimientos

Agradezco a dios y a mi familia por acompañarme durante el desarrollo de mi vida, sobre todo en el cumplimiento de esta meta de mi vida, y permitirme siempre sentirme apoyado en cada momento de fragilidad, ustedes siempre serán mi apoyo.
Un sincero agradecimiento a la Universidad Distrital Francisco José de Caldas por brindarme los espacios de aprendizaje, al ingeniero Juan Pablo, a los doctores Giovanny Tarazona y Sandro Javier Bolaños de la facultad de ingeniería, al doctor Vicente García Díaz de la Universidad de Oviedo-España, y a mi compañero John Petearson Anzola, por acompañarme en este proceso y brindarme una mano amiga para sacar adelante este proyecto.
A la Fundación Universitaria los Libertadores, le estoy inmensamente agradecido por darme la oportunidad de culminar esta etapa de mi vida, y por apoyarme de la misma manera para poderlo realizar.

Dedicatoria

A Dios, mi familia y amigos.

Resumen

Los Sistemas Multi-Agentes Robóticos Autónomos (MARS) son modelados y desarrollados para trabajar en procesos dedicados a la navegación en entornos estocásticos, especialmente en localización, planeación y mapeo. Estos procesos hacen uso de modelos de diseño en sistemas centralizados que generan dependencia de una unidad central, siendo esto una limitante para la autonomía de los agentes robóticos. Es por esto que la descentralización de un sistema multi-agente robótico ha sido un área de investigación en los últimos años, ya que permite el control y asignación de tareas distribuidas de manera asincrónica y robusta al no tener la dependencia de una unidad central. Sin embargo, hay áreas en las que no se contempla el proceso de comunicación entre la red generada por los agentes robóticos, creando dependencia en una conexión permanente a las unidades vecinas para realizar los objetivos propuestos.

En este trabajo de investigación se describe un modelo descentralizado, que permite la interacción entre los agentes robóticos que componen el sistema y agentes externos, de manera descentralizada, basando su arquitectura de funcionamiento en la tecnología emergente de las as redes de sensores inalámbricas (WSN).

La implementación del modelo descentralizado propuesto hace uso de un framework de comunicación, en donde, cada agente que compone el sistema puede abandonar la red o aceptar nuevas conexiones, enviando la información necesaria basándose en el historial de transferencia entre todos los nodos que compongan la red. Para lograr esto, cada agente debe cumplir con cinco procesos; los cuatro procesos iniciales permiten el ingreso de un agente al sistema, y el quinto proceso, permite que el agente envíe y reciba información de los agentes más cercanos basado en el Received Signal Strength Indicator (RSSI) y el historial de datos.

Para la validación del modelo descentralizado y del framework, se hace uso de agentes robóticos diferenciales y agentes de monitoreo, los cuales son ubicados en un entorno desconocido, y como objetivo deben realizar tareas de navegación tales como la estimación de la posición y la generación del mapa de manera cooperativa, demostrando la fiabilidad del modelo en tareas de navegación de manera descentralizada.

Palabras clave: Agentes móviles, Control descentralizado, Modelelamiento, Sistema multi-agente, Redes de sensores inalámbricas.)

Abstract

The autonomous Multi-Agent Robotic Systems (MARS) are modeled and developed to work in processes dedicated to navigation in stochastic environments, especially in localization, planning, and mapping. These processes make use of design models in centralized systems that generate dependence of a central unit, being this a constraint for the autonomy of the robotic agents.

This is why the decentralization of a robotic multi-agent system has been an area of research in recent years since it allows the control and allocation of distributed tasks in an asynchronous and robust manner since it does not have the dependence on a central unit. However, there are areas in which the communication process in the network generated by the robotic agents is not contemplated, creating dependency on a permanent connection to the neighboring units to realize the proposed objectives.

In this research project, a decentralized model is described, which allows interaction not only between the robotic agents that make up the system, but also, with external agents such as monitoring units or sensors, in a decentralized manner, basing its operating architecture on the emerging technology of Wireless Sensor Networks (WSN).

The implementation of the proposed decentralized model makes use of a communication framework, where each agent that composes the system can leave the network or accept new connections, sending the necessary information based on the transfer history between all the nodes that make up the network. To accomplish this, each agent must comply with four processes to be able to belong to the system, and a fifth process for sending information to the closest nodes based on the Received Signal Strength Indicator (RSSI) and the data history.

For the validation of the decentralized model and the framework, are used differential robotic agents and monitoring agents, which are located in an unknown environment, and as an objective they must perform navigation tasks such as the estimation of the position and the map generation cooperatively, demonstrating the reliability of the model in navigation tasks in a decentralized way.

Keywords: Decentralized control, Modeling, Mobile agents, Multi-agent systems, Wireless sensor networks.)

Contenido

Agradecimientos I

Dedicatoria II

Resumen III

Abstract IV

I CONTEXTO 1

1. CAPÍTULO 1: FUNDAMENTO DE LA INVESTIGACIÓN 2

1.1. INTRODUCCIÓN 2

1.2. JUSTIFICACIÓN DE LA INVESTIGACIÓN 4

1.2.1 Justificación Tecnológica 5

1.2.2 Justificación Socio-Ambiental 5

1.2.3 Justificación Ambiental 6

1.3. PROBLEMÁTICA 6

1.3.1 DESCRIPCIÓN DEL PROBLEMA 6

1.3.2 FORMULACIÓN DEL PROBLEMA 7

1.4. HIPÓTESIS DE LA INVESTIGACIÓN 8

1.5. OBJETIVOS DE LA INVESTIGACIÓN 8

1.5.1 OBJETIVO GENERAL 8

1.5.2 OBJETIVOS ESPECÍFICOS 8

1.5.3 PREGUNTA DE LA INVESTIGACIÓN 8

1.5.4 METODOLOGÍA 9

1.6. ESTRUCTURA DE LA TESIS 10

2. CAPÍTULO 2: MARCO TEÓRICO 12

2.1. ANTECEDENTES 12
2.2. SISTEMAS ROBÓTICOS 14
2.3. SISTEMAS ROBÓTICOS ASISTENCIALES 15
2.4. SISTEMAS MULTI-AGENTE (MAS) 17
2.5. SISTEMAS MULTI-AGENTE ROBÓTICOS (MARS) 18
2.6. REDES DE SENSORES INALÁMBRICAS 19
2.7. ALGORITMOS DE CONSENSO 20

II DESARROLLO 23

3. CAPÍTULO 3: MODELO DE INTEGRACIÓN BASADO EN EL CONCEPTO DE WSN 24

3.1. INTRODUCCIÓN 24
3.2. APLICACIÓN DE LAS WSN EN LOS SISTEMAS MULTI-AGENTE . . . 24
3.3. MODELO DEL SISTEMA 27
3.4. MODELO DE COMUNICACIÓN BASE ENTRE LOS AGENTES 29
3.5. EVALUACIÓN DEL MODELO 32

4. CAPÍTULO 4: APLICACIÓN DEL MODELO EN SISTEMAS ROBÓTICOS MULTI-AGENTE 34

4.1. INTRODUCCIÓN 34
4.2. ARQUITECTURA DEL SISTEMA 34
4.2.1 Descripción de los agentes 36
4.2.2 Puesta de los sensores e intercambio de la información 38
4.3. PRUEBA DEL SISTEMA 43

5. CAPÍTULO 5: FRAMEWORK PARA LA DESCENTRALIZACIÓN DE SISTEMAS ROBÓTICOS MULTI-AGENTE 48

5.1. INTRODUCCIÓN 48
5.2. DISEÑO DEL FRAMEWORK 48
5.2.1 Descripción de los agentes 49
5.2.2 Descripción del framework 52

5.3. APLICACIÓN DEL FRAMEWORK PARA MARS 61

III REFLEXIONES 69

6. CAPÍTULO 6: CONCLUSIONES 70

7. CAPÍTULO 7: CORRESPONDENCIA HIPÓTESIS - RESULTADOS 71

8. CAPÍTULO 8: APORTES Y DIVULGACIONES 74
8.1. APORTES 74
8.2. REVISTAS INDEXADAS 75
8.3. EVENTOS CIENTÍFICOS 75

9. CAPÍTULO 9: TRABAJOS FUTUROS 76

Bibliografía 77

A. GLOSARIO 87

B. Anexo: Diagrama de flujo algoritmo 1 88

C. Anexo: Diagrama de flujo algoritmo 2 89

D. Anexo: Diagrama de flujo algoritmo 3 90

E. Anexo: Diagrama de flujo algoritmo 4 91

F. Anexo: Código control y monitoreo agente robótico 92

G. Anexo: Código agente de monitoreo y de control 106

H. Anexo: Evaluación módulos de comunicación 119

I. Anexo: Simulación V-Rep generación de mapa 121

Lista de Figuras

1-1 Número de ventas de agentes robóticos asistenciales entre los años 2015-2016, y proyección al año 2020 4
1-2 Metodología de investigación empleada. 9

2-1 Navegación de un robot por medio de WSN. 12
2-2 Robots conectados con los vecinos más cercanos en el rango de comunicación 13
2-3 Arquitectura de comunicación en enjambres robóticos 14
2-4 Brazo robótico de ST Robotics 16
2-5 Silla usada por Stephen Hawking 16
2-6 Aspiradora robot de Irobot 17
2-7 Robot guía RHINO 17
2-8 Modelo de un agente con reglas simples 18
2-9 Modelo de un agente con reglas de aprendizaje 18
2-10 Modelo básico agente robótico 19
2-11 Esquema Red de Sensores Inalámbricas 20
2-12 Clúster de servidores usados junto con los clientes 21
2-13 Arquitectura de la máquina de estados replicada 21

3-1 Mapa mental de autonomia en agentes robóticos 25
3-2 Diagrama de casos de uso del diseño básico del agente en WSN centralizado 28
3-3 Diagrama de secuencia para el diseño del agente robótico descentralizado con WSN 28
3-4 Diagrama de trabajo comunicación y procesamiento del agente robótico . . 29
3-5 (a) Agentes distribuidos sin conexión. (b) Conexión entre los agentes basándose en el RSSI (c) Conexión entre los agentes basándose en el RSSI con un obstáculo 31

4-1 Asignación de identificadores del AR respecto el plano cartesiano. 36
4-2 Ubicación de los sensores de proximidad. 37

4-3 Actualización arreglo del AR (**a**) posición inicial; (**b**) agregando un nuevo nodo; (**c**) actualizando caminos pendientes. 39
4-4 Intercambio de información entre dos agentes. 42
4-5 Evaluación del modulo wi-fi ESP8266 (**a**) paquetes perdidos; (**b**) retraso promedio; (**c**) relación distancia - RSSI. 44
4-6 Entorno de pruebas. 45
4-7 Resultados del experimento usando un sistema MAS (**a**) puesta de los nodos WA; (**b**) mapa topológico 45
4-8 Puesta de los sensores en un sistema centralizado. 46
4-9 Inyección de un error al RA (**a**) puesta de los nodos WA; (**b**) mapa topológico 47

5-1 Diagrama del agente robótico. 49
5-2 Modelo dinámico de un robot diferencial de dos ruedas. 50
5-3 Diagrama del agente inalámbrico o unidad de monitoreo. 51
5-4 Generación de la matriz de puntos por odometría. 53
5-5 Análisis transmisión de datos (**a**) retardo de la información; (**b**) indicador nivel RSSI 55
5-6 Análisis transmisión de datos (**a**) Transferencia de información por pedido; (**b**) Envió de información de forma inmediata 60
5-7 Experimento con dos agentes robóticos. 63
5-8 Mapa topológico procesado por dos agentes robóticos. 63
5-9 Experimento con tres agentes robóticos. 65
5-10 Mapa topológico procesado por dos agentes robóticos. 65
5-11 Mapa topológico procesado por dos agentes robóticos. 66
5-12 Experimento con tres agentes robóticos y un agente de monitoreo. 68

Lista de Tablas

3-1 Resultados del modelo aplicado en un sistema MAS frente a un sistema centralizado. 32
3-2 Resultados del modelo aplicado en un sistema MARS frente a un sistema centralizado. 33

4-1 Arreglo dinámico de memoria para el agente inalámbrico. 38
4-2 Arreglo dinámico de memoria para el agente robótico. 38
4-3 Actualización del arreglo dinámico del AR 40
4-4 Inicialización de los arreglos Fig. **4-4**. 42
4-5 Actualización de los arreglos Fig. **4-4** después del intercambio de información. 42

5-1 Arreglo dinámico del historial de información. 51
5-2 Arreglo dinámico del historial de información. 51
5-3 Arreglo dinámico del historial de información. 52
5-4 Arreglo dinámico del historial de información. 56
5-5 Inicialización del historial con 3 agentes. 57
5-6 Transferencia de información entre los agentes Ro_1 y Ro_2. 57
5-7 Transferencia de información entre los agentes Ro_2 y Ro_3. 58
5-8 Transferencia de información entre los agentes Ro_2 y Ro_1, enviando la información recibida del agente Ro_3 59
5-9 Códigos de identificación para la transmisión de información entre los agentes. 59
5-10 Resultados del experimento con dos agentes robóticos. 64
5-11 Resultados del experimento con tres agentes robóticos. 66
5-12 Resultados del experimento con tres agentes robóticos después de la falla del agente $robot_3$. 67
5-13 Resultado del experimento con tres agentes robóticos y una unidad de monitoreo externa. 68

8-1 Revistas indexadas. 75
8-2 Participación en ponencias internacionales. 75

Parte I.

CONTEXTO

1. CAPÍTULO 1: FUNDAMENTO DE LA INVESTIGACIÓN

1.1. INTRODUCCIÓN

En este trabajo de investigación se presenta un modelo de descentralizado para sistemas robóticos multi-agente, orientado al apoyo de tareas de navegación cooperativas, el cual permite aumentar la autonomía evitando la dependencia de unidades externas para el procesamiento y control de las tareas de los agentes que componen el sistema.

Se ha demostrado que un sistema compuesto por múltiples agentes robóticos (MARS) es capaz de resolver tareas de búsqueda y rescate de personas en entornos de difícil acceso [53], vigilancia y monitoreo de edificios [25], puesta y medición de sensores en entornos desconocidos de manera más eficiente que un solo agente robótico para captura o generación de mapas [76, 77]. Para el desarrollo de estas tareas, los MARS deben tener una estructura de control distribuido entre todas las unidades, de manera que cada agente que pertenece al sistema sea autónomo al no depender de manera directa de un operador humano para su funcionamiento, siendo esto posible por la cooperación entre todos los agentes que comparten la información por medio de una red.

La estructura de control distribuido puede realizarse de forma centralizada o descentralizada, en el caso de un modelo centralizado, el sistema depende de una unidad central para el procesamiento de la información requiriendo toda la información para la toma de decisiones, mientras en una estructura descentralizada, el procesamiento se da por medio de la información local de cada agente del MARS [37, 4]. Furno et al. [28] compara estos dos tipos de estructuras con un sistema compuesto por tres agentes robóticos submarinos para tareas de transporte de equipo; el sistema centralizado, muestra un menor tiempo para corregir errores entre los actuadores para controlar la dirección y la velocidad, pero el sistema descentralizado presenta mejores características al tolerar fallas en el sistema por parte de un agente robótico, además, debido a la limitante del ancho de banda en el canal de comunicaciones, reducen el tamaño del paquete de información a transmitir entre las unidades, pero el envió de la información debe ser permanente entre las tres unidades, tanto en el modelo centralizado como en el descentralizado.

Esta tolerancia a fallos en los sistemas de control distribuido descentralizados es aprovechada

en diferentes áreas, Schwager et Al. [68], propone la distribución de sensores usando un MARS donde cada agente comparte la informatización solo con el vecino más próximo dentro de un área convexa, es decir, no se contempla la discontinuidad en el entorno, sea por agujeros o por obstáculos en el entorno o área de trabajo. Cheng et. Al. [14], realiza un sistema robusto, que permite realizar tareas de reconocimiento y captura de información, por medio de sensores en un área predeterminada, manteniendo una comunicación permanente entre todos los agentes, para eliminar fallas con respecto al procesamiento de la velocidad y la dirección que cada robot móvil.

Otro tipo de aplicación es la formación de robots móviles usando roles y topologías de redes específicas para optimizar la comunicación en la red, siendo un modelo multi-agente descentralizado por enviar solo la información entre nodos vecinos, pero centralizan la operaciones de funcionamiento al depender de un líder en la formación, o en el caso de ser un sistema adaptativo, comparten una función de costo-recompensa para modificar el sistema [72, 13, 3]. Otra característica muy importante de los MARS, es su capacidad de ser heterogéneos, es decir pueden estar compuestos de diferentes unidades, Tanner en su trabajo [74] demuestra que los MARS pueden ser un sistema heterogéneo y descentralizado al estar compuestos por Unmanned Aerial Vehicles (UAV) y Unmanned Ground Vehicles (UAV), donde se resalta que es descentralizado al no depender de una unidad central, pero es dependiente de tener una comunicación permanente con una topología especifica.

El problema de la dependencia de una topología específica y de tener siempre una comunicación permanente entre las unidades o una unidad central, puede ser corregido al hacer uso de redes Ad-Hoc, Ming et al. [70] propone un sistema autónomo de exploración usando MARS, demostrando que los agentes que se encuentran en el sistema pueden estar conectados o desconectados en cualquier momento evitando la permanente conexión entre los agentes, pero no muestran un estudio para evitar la pérdida de paquetes y la posible conexión con nuevos agentes para complementar el sistema en cualquier momento.

Para evitar la dependencia de topologías especificas en los agentes de un sistema, el modelo presentado en este trabajo de investigación permite la descentralizado de los sistemas MARS, eliminando el requerimiento de comunicación permanente entre los agentes-robóticos y agentes-inalámbricos, integrando la conexión de unidades externas para apoyar el sistema, sea para monitoreo o completar la tarea asignada inicialmente de navegación de manera descentralizada.

El desarrollo del modelo se realiza en tres fases. En la primera, se plantean los requerimientos que deben tener las unidades que componen al sistema, haciendo uso del concepto de las redes de sensores inalámbricas. En la segunda fase, el modelo es validado en un entorno

de simulaciones desarrollado por *coppelia robotics*[1], usando agentes robóticos diferenciales en tareas de navegación. Por último, en la tercera fase se construye un framework para la aplicación del modelo, el cual permite agregar agentes heterogéneos al sistema, extendiendo la capacidad integración con unidades externas al sistema en tareas de navegación.

1.2. JUSTIFICACIÓN DE LA INVESTIGACIÓN

Los agentes robóticos han incrementado su popularidad en los últimos años, apoyando a los seres humanos desde tareas industriales hasta domesticas. Los sistemas roboticos multi-agente (MARS), han demostrado ser más eficientes en el cumplimiento de objetivos, al usar un equipo completo de robots al lugar de una simple unidad robótica. Estos objetivos incluyen la resolución de problemas orientados a la robótica asistencial, tales como tareas de de limpieza, mantenimiento, construcción y exploración de entornos desconocidos.

A nivel mundial para el año 2020 se proyecta un incremento apreciable en el uso de los agentes robóticos Fig. (**1-1**). En el Sector medico, se proyecta un incremento de un 435 %; para el sector de la construcción se proyecta un incremento del 300 %; el uso de exoesqueletos para el sector industrial se proyecta un incremento del 406.17 %, y para el sector de relaciones publicas, que incluye el uso de agentes robóticos móviles como guías turísticos, se proyecta un incremento del 541.74 %.

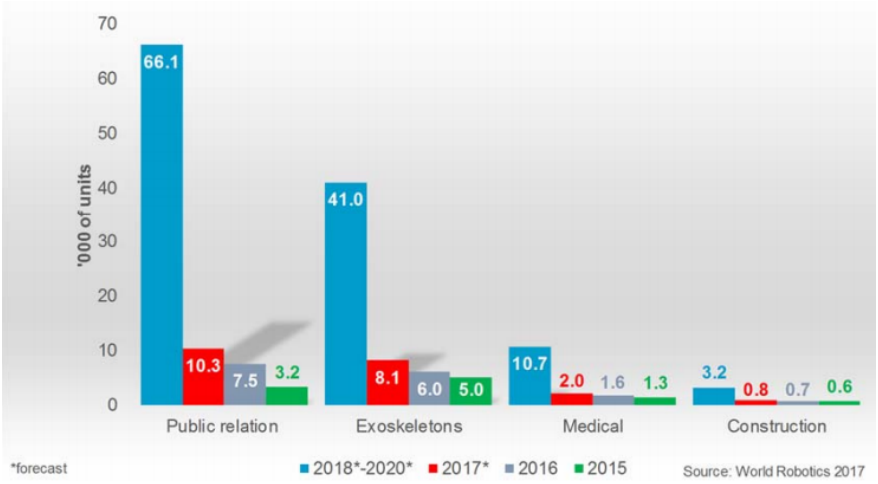


Figura 1-1.: Número de ventas de agentes robóticos asistenciales entre los años 2015-2016, y proyección al año 2020

Fuente: `https://ifr.org/`

[1]Disponible en: `http://www.coppeliarobotics.com/`

1.2.1. Justificación Tecnológica

En los MARS, la distribución y el procesamiento de información entre los agentes, se caracteriza por ser autónoma con una alta tolerancia a fallos al ser modular y tener una arquitectura distribuida [26, 55]. La modularidad en el sistema permite que un sistema sea robusto, al poder detectar y reemplaza con facilidad agentes o partes de estos que no estén funcionando, de igual forma, si el sistema necesita ser actualizado parcialmente, solo se cambian los agentes necesarios, reduciendo el costo de mantenimiento [5, 75], y siendo un sistema distribuido, pueden realizar múltiples tareas al mismo tiempo de manera asincrónica sin depender de un control global [56].

Para la comunicación entre los agentes, se han usado enfoques con tecnologías de Wireless Sensor Network (WSN) por estar compuestas por nodos con unidades independientes de procesamiento, módulos de comunicación inalámbricos y sensores, los cuales pueden ser estáticos o móviles como en el caso de los agentes robóticos [48, 50].

El uso de esta tecnología ha impactado las aplicaciones de los MARS. García et. al. [30] hace el uso de las WSN para recibir información de múltiples agentes, para aumentar la eficiencia en el uso de recursos y administrar los sistemas de transporte en un aeropuerto de carga, en este caso, los agentes robóticos son actuadores estáticos, que, aunque compartan información de manera local, dependen del procesamiento de la información por un sistema de control único.

En este trabajo de investigación, se busca extender el impacto de las aplicaciones de los MARS, al eliminar la dependencia del uso de unidades centrales. Para esto, se plantea un modelo descentralizado que puede ser aplicado a sistemas multi-agentes (MAS) o MARS, que aumente la autonomía de todo el sistema. El modelo propuesto se orienta a tareas de navegación usando agentes robóticos, lo cual, permitiría ser aplicado a la coordinación de múltiples agentes en entornos de emergencia

1.2.2. Justificación Socio-Ambiental

El propósito de este trabajo de investigación es generar un modelo descentralizado para realizar tareas de navegación haciendo uso de MARS, el cual, se puede aplicar a los diferentes niveles del área de trabajo de alimentos funcionales y naturales bajo el proyecto "Especialización Inteligente Bogotá-Región"[2], siendo el objetivo de esta área la presentación de servicios relacionados con los procesos de: conocimiento, de negocios y de servicios informáticos.

Por lo anterior, este modelo puede impactar directamente en la automatización de procesos

[2]Información referida de `http://www.ccb.org.co/Transformar-Bogota/Especializacion-Inteligente-Bogota-Region/Areas-de-especializacion`

usando un sistema multi-agente robótico, tales como la generación de estrategias de riego y de monitoreo de los cultivos en tiempo real, sin hacer uso de infraestructuras de comunicación fijas al ser un sistema descentralizando, generando un valor agregado en las etapas de cultivo y recolección.

1.2.3. Justificación Ambiental

En el libro verde 2030 del gobierno de Colombia [3], se establece la importancia de la innovación ambiental sostenible, tales como el apoyo al fomento para el desarrollo de tecnologías limpias, como por ejemplo el cambio de energía fósil por el uso de la energía solar, siendo uno de los Objetivos de Desarrollo Sostenible (ODS 15).
El uso del modelo propuesto podría extender su uso, para que los MARS puedan ser usados en las plantas productoras de energía solar, permitiendo reducir la actividad antrópica haciendo más eficiente las labores de producción y monitoreo.

1.3. PROBLEMÁTICA

1.3.1. DESCRIPCIÓN DEL PROBLEMA

Dada la complejidad del diseño de los Sistemas MultiAgentes Robóticos (MARS) en ambientes estocásticos, y su alta demanda en los últimos años (para el 2020 se estima 32.4 millones de robots empleados a nivel mundial [31]), se han convertido en un área multi-disciplinar, haciendo necesaria una estructura de diseño basada en modelos aplicados a la programación por objetivos, la arquitectura del robot y la administración de tareas [6].
En los MARS los problemas son modelados como una sola unidad, siendo útil cuando los sistemas no son complejos, pero al tener en cuenta el comportamiento dinámico y estocástico de los ambientes se hace necesario el uso de modelos que ayuden a representar los problemas de navegación en sub-tareas, tal como la planeación, localización y mapeo, ello significa comprender el entorno, conocer los recursos disponibles a nivel de hardware y las herramientas que permita procesar y modelar el entorno a nivel de software, y trabajar el problema de manera distribuida sin obviar la relación entre los agentes. Para lograr lo anterior, se aplican estrategias que hacen uso de dispositivos de seguimiento óptico (OTD), visión estereoscópica, técnicas de visión artificial con algoritmos inteligentes y redes de sensores inalámbricos (WSN) [47, 83, 49, 38, 33, 57].

[3] Libro disponible digital en: `http://files.constantcontact.com/b884f834401/7cd9f0a0-433d-4d55-9f24-5c8ffc82273b.pdf`

En las tareas de navegación, se busca aumentar la autonomía en cada uno de los agentes que integra el sistema, esto es, cada agente debe tener la capacidad de procesar los datos de manera distribuida y descentralizada, también debe tener un sistema operativo que garantice la administración de tareas de procesamiento del sistema en tiempo real, sin olvidar las propiedades de tener una arquitectura híbrida (integración software-hardware), la generación de modelos o patrones, la capacidad de aprendizaje y la modularidad del sistema [75, 56, 86]. Al tener múltiples agentes en la resolución de problemas de navegación, se reduce el tiempo para alcanzar el objetivo al aumentar la capacidad de procesamiento, sin embargo se hace más difícil la administración de la autonomía en el sistema, en especial el control entre la interacción de los agentes, debido a que los agentes deben comunicarse de manera inalámbrica para intercambiar o procesar la información de manera distribuida, sacrificando la des-centralización del sistema para reducir la complejidad. La tecnología que se usa para este intercambio de información puede ser vista como una Red de Sensores Inalámbricas WSN, debido que cada agente puede ser visto como un nodo en el sistema [62, 12, 15, 29, 73, 84]. A comparación de las otras estrategias, la que aplica tecnología en WSN, permite ubicar a los MARS entornos no controlados por medio de sensores distribuidos en grillas de nodos estáticos o móviles, enviando la información hacia una unidad central encargada de realizar las tareas de procesamiento, reduciendo la complejidad computacional al permitir dividir las tareas de computo [48]. Esta tecnología se caracteriza por su flexibilidad permitiendo el mapeo y localización del entorno mediante la lectura del parámetro de indicador de fuerza de la señal recibida (RSSI) aplicando técnicas de fusión de información entre los diferentes agentes [85, 10, 9, 65]. Sin embargo, las WSN requieren de una unidad central que procese la información y controle la comunicación del sistema, reduciendo la autonomía del sistema al depender de una unidad externa para su funcionamiento.

1.3.2. FORMULACIÓN DEL PROBLEMA

Al tener una unidad central para el procesamiento de la información, se presenta un problema de dependencia del sistema para su funcionamiento, ya que si la unidad central llegara a presentar fallas el sistema colapsaría. Por otro lado, el sistema siempre debe mantener comunicación con la unidad central, de esta manera, todos los nodos deben comunicarse hacia la unidad central, necesitando de rutas indirectas de comunicación si la unidad central se encuentra fuera del radio de cobertura, aumentando el consumo de energía y la complejidad computacional al implementar algoritmos de enrutamientos sobre los nodos [18].
De esta manera, se hace necesario eliminar la dependencia de unidades externas, y además, evitar que todos los agentes del sistema se estén comunicación de forma permanente, para el funcionamiento de los MARS en tareas de navegación. Para lograr esto, es necesario

desarrollar un modelo descentralizado y distribuido para los MARS, orientado a tareas de navegación usando un nuevo algoritmo de consenso evitando la relación servidor/cliente.

1.4. HIPÓTESIS DE LA INVESTIGACIÓN

La generación de modelos descentralizados en sistemas robóticos cooperativos basados en la tecnología WSN, permite desarrollar sistemas capaces de realizar tareas de navegación en entornos dinámicos y estocásticos evitando el uso de una unidad central.

1.5. OBJETIVOS DE LA INVESTIGACIÓN

1.5.1. OBJETIVO GENERAL

Modelar los sistemas robóticos multi-agentes cooperativos para tareas de navegación descentralizada en entornos dinámicos y estocásticos.

1.5.2. OBJETIVOS ESPECÍFICOS

- Generar un modelo de integración que establezca la relación entre los recursos y el procesamiento, a partir del concepto de WSN.
- Validar el modelo de integración en MARS en robots cooperativos.
- Analizar el comportamiento de los agentes robóticos en ambientes dinámicos y estocásticos al usar el modelo de integración.
- Diseñar un algoritmo de consenso descentralizado para agentes robóticos orientado a realizar tareas de navegación eliminando el concepto de cliente/servidor.

1.5.3. PREGUNTA DE LA INVESTIGACIÓN

¿Cuáles elementos de los sistemas robóticos multi-agentes permiten aumentar la autonomía del agente para desarrollar tareas de navegación cooperativa en entornos dinámicos y estocásticos?

1.5.4. METODOLOGÍA

La metodología de investigación empleada para el proyecto tiene como base el método científico, en donde se divide el proyecto en 5 etapas, haciendo uso de la Ingeniería de Software, Sistemas Digitales, Sistemas Operativos y Teoría de sistemas dinámicos como se muestra a continuación (Fig. **1-2**):

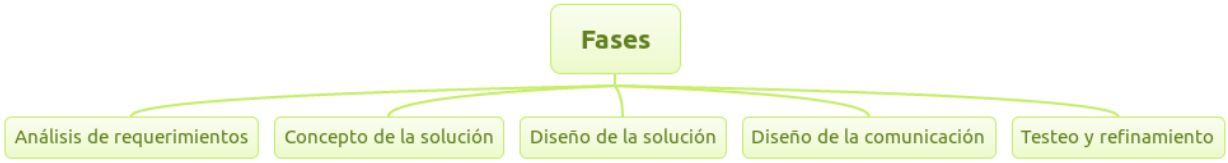


Figura 1-2.: Metodología de investigación empleada.
Fuente: Propia

1. **Análisis de requerimientos:** se analiza el ambiente y las tareas que debe desempeñar el MARS con algoritmos cooperativos de planeación, localización y mapeo.
   a) Modelar los algoritmos en lenguajes orientados a objetos.
   b) Modelar los entornos de navegación de los MARS en dos dimensiones.
2. **Concepto de la solución:** se generan alternativas de los sensores y los actuadores del MARS, los cuales son evaluados para posteriormente crear un análisis de consumo de energía, capacidad de cómputo y tiempo de procesamiento, donde los sensores y actuadores deben ser capaces de capturar y modificar los requerimientos mínimos del ambiente para realizar la tareas de navegación.
   a) Simular el comportamiento de los sensores.
   b) Comparar las simulaciones con las características tomadas por un laboratorio.
   c) Generar una plataforma de emulación con los resultados.
3. **Diseño de la solución:** De acuerdo a la integración de la unidad de control y la tarea que el MARS debe resolver, se hace un estudio del algoritmo de navegación.
   a) Selección de la unidad central del agente robótico.
   b) Verificar el funcionamiento de la unidad central con los sensores y los actuadores seleccionados.
4. **Diseño de comunicación:** Se desarrolla un estudio para generar alternativas en los módulos de comunicación los cuales pueden ser Wifi, bluetooth o Zigbe.

a) Realizar experimentos de comunicación del agente con un computador.

b) Realizar tareas de navegación domando muestras del agente para analizar su comportamiento.

c) Crear un sistema capaz de simular el comportamiento de la comunicación del ambiente.

d) Diseño del algoritmo descentralizado.

e) Diseño de un simulador por medio de un motor de juegos capaz de simular múltiples agentes.

f) Modelar el comportamiento de una red inalámbrica en el simulador.

5. **Testeo y refinamiento:** Construcción y evaluación de múltiples versiones del producto, incluyendo prototipos.

a) Evaluación del algoritmo de comunicación descentralizado.

b) Pruebas en el simulador del desempeño del algoritmo con múltiples agentes.

c) Realizar experimentos y comparar los resultados con diseños centralizados.

1.6. ESTRUCTURA DE LA TESIS

El presente trabajo de investigación se encuentra dividido en los siguientes capítulos:

- En el **capítulo 1** se expone la problemática y la justificación de la investigación, la hipótesis y los objetivos que permitieron el desarrollo del proyecto.

- El **capítulo 2** presenta los antecedentes de los sistemas robóticos multi-agente orientados a tareas de navegación, junto con una breve base teórica de los sistemas MARS, las redes de sensores inalámbricas y los algoritmos de consenso.

- El modelo descentralizado es presentado en el **capítulo 3**, el cual se basa en la integración de los sistemas MARS con la flexibilidad y modularidad de las redes de sensores inalámbricas.

- La aplicación del modelo descentralizado en tareas de navegación es presentado en el **capítulo 4**, permitiendo la localización de un agente robótico y la generación simultanea de un mapa topológico usando un sistema compuesto por agentes inalámbricos y un agente robótico.

- Para facilitar el uso del modelo, en el **capítulo 5** se presenta un framework que permite la implementación de agentes heterogéneos, es decir, el sistema MARS también puede aceptar otros agentes externos al sistema en cualquier momento para apoyar o monitorear las tareas de navegación.

- En el **capítulo 6**, se presentan las conclusiones generales de este trabajo de investigación.

- En el **capítulo 7**, se hace un contraste entre los objetivos propuestos y los resultados obtenidos en este trabajo de investigación.

- Finalmente, en el **capítulo 8**, se presentan los aportes junto con un listado de las publicaciones y ponencias realizadas durante el desarrollo de este trabajo de investigación.

- En el **capítulo 9**, se presentan sugerencias para trabajos futuros.

2. CAPÍTULO 2: MARCO TEÓRICO

2.1. ANTECEDENTES

Los agentes robóticos que hacen uso de WSN para realizar tareas de navegación usan nodos de sensores ubicados espacialmente en el entorno a modo de guías estáticas, siendo estas usadas como puntos de localización al tener a priori la posición exacta donde son instalados; estos sistemas permiten tener un bajo nivel de cómputo para procesar la informática, dado que solo necesitan tener como referencia de la RSSI de cada uno de los nodos para ubicarse, sin necesidad de recurrir a la captura de imágenes como lo requieren las técnicas de OTD como se muestra en la Fig. **2-1** [85, 22].

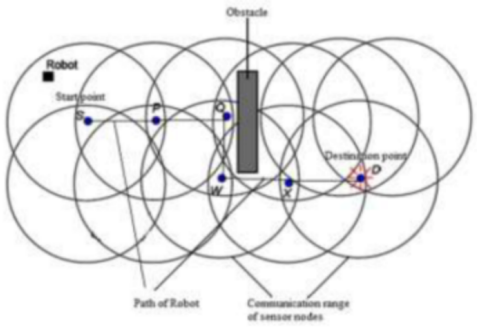


Figura 2-1.: Navegación de un robot por medio de WSN.
Fuente: [78]

En las WSN también se ha usado el concepto de autonomía, siendo este orientado a trabajar con Sistemas Multi-Agentes (MAS), describiendo cada nodo de sensado como una tecnología capaz de ser ubicada en cualquier entorno, percibiendo las características de este (dependiendo de los sensores), y basado en la dinámica del entorno se ejecutan planes de acción pre programados para interactuar directamente por medio de actuadores, o se envían señales a la unidad central para la toma de decisiones [48]. Estos planes de acción están orientados a tareas de auto-mantenimiento, auto-protección, auto-optimización y auto-configuración, con el fin de reducir recursos de cómputo y consumo de energía, siendo un excelente candidato para ser empleadas de forma directa en agentes móviles como los agentes robóticos orientados a tareas de navegación [78].

Otros agentes robóticos trabajan en dos fases la navegación usando WSN como referente, la primera fase consiste en adquirir la información de los nodos de sensores para realizar una estimación por medio de filtros de Bayes sin conocer a priori la localización de los sensores, modelando la localización de estos por medio de una distribución de probabilidad. La posición del agente robótico es incluida en esta fase con el fin de reducir el problema de la distorsión de la RSSI. En la segunda fase, la red puede computar mejor la estimación de posición usando un filtro de información para computar la localización del agente en el entorno de la WSN, repitiendo este proceso hasta encontrar la posición exacta tanto de los sensores como del agente robótico [8].

En el desarrollo del esquema de control descentralizado, se usan técnicas y protocolos en la comunicación para controlar el tráfico de información de manera óptima, aplicando algoritmos de consenso; estos se basan en la comunicación permanente entre servidores dentro de un clúster para evitar fallas en la redes al compartir la información de manera redundante entre todos los nodos. El esquema de control descentralizado usando algoritmos de consenso ha sido implementado en la robótica cooperativa para evasión de obstáculos, como se evidencia en el trabajo realizado por Kwon *et al.* [39] y en la Fig. **2-2**, donde la conexión puede hacerse solamente con los agentes vecinos que se encuentren en el rango de comunicaciones, seleccionando de manera aleatoria un líder para administrar la transferencia de datos, siendo usada para actualizar la información del entorno en todos los agentes, y se completa este proceso después de cierto tiempo debido al problema de sincronización en los relojes de los agentes, siendo un inconveniente en el consumo de energía, además siendo el caso que el problema se resuelva de manera distribuida y descentralizada, la sincronización afecta los estados en los lideres [42, 61, 52, 11, 46, 82].

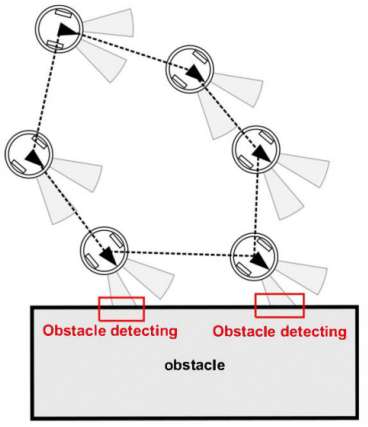


Figura 2-2.: Robots conectados con los vecinos más cercanos en el rango de comunicación
Fuente: [39]

Otro tema interesante es la vigilancia automática por medio de colonia de robots o MARS, estos trabajos se caracterizan por manejar un ambiente estocástico donde los agentes deben compartir información relevante del entorno para identificar posibles amenazas en el entorno. Este tipo de sistemas es muy apetecido por su bajo nivel de procesamiento y su alta tolerancia a la falla, dado que si un agente es lastimado por alguna amenaza, se puede detectar de forma temprana o simplemente el agente se puede reemplazar por otro, topologías usadas para resolver este problema se evidencia en la Fig. **2-3**, donde los MARS son distribuidos en una malla de sensores inalámbricos comunicándose con el nodo más cercano para transmitir información a la unidad central de control [44].

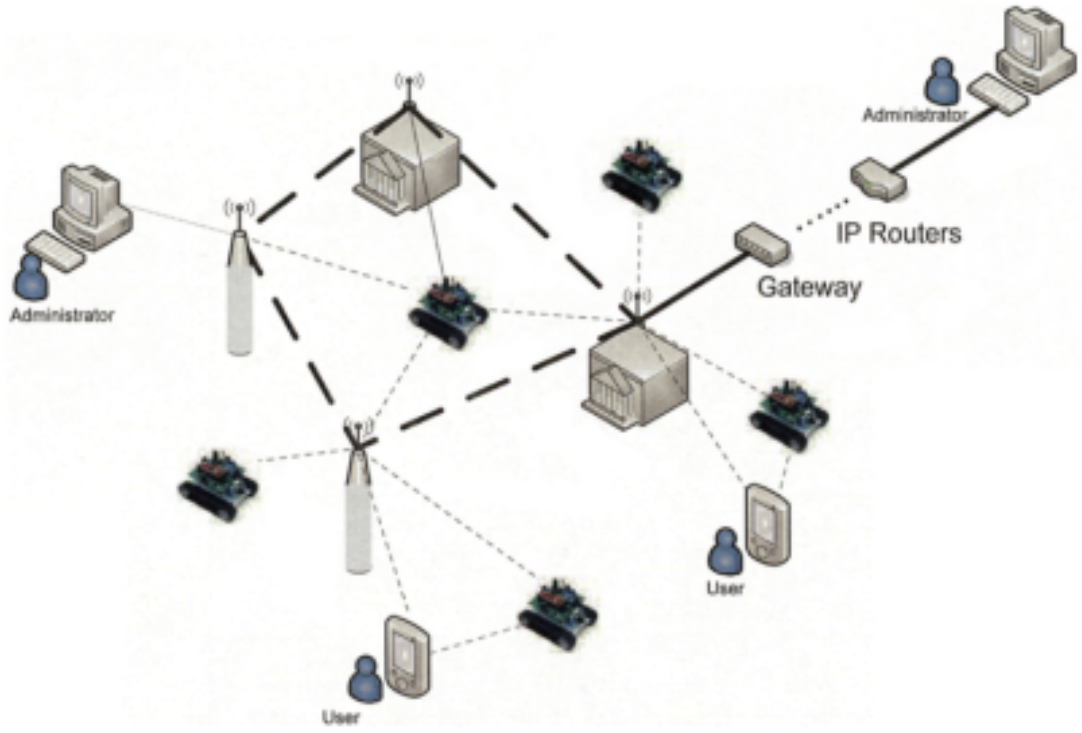


Figura 2-3.: Arquitectura de comunicación en enjambres robóticos
Fuente: [44]

2.2. SISTEMAS ROBÓTICOS

Consiste en un sistema compuesto por múltiples robots, los cuales pueden ser re configurados por un operario para realizar diversas tareas de automatización en procesos de fabricación. Los robots que componen el sistema no necesariamente buscan cumplir un mismo objetivo, más sin embargo, estas pueden pertenecer a la misma red de comunicaciones [51].

Los sistemas robóticos se pueden estar compuestos por tres diferentes sub-sistemas, los sistemas de manipulación, sistemas móviles y sistemas de adquisición.

Los sistemas de manipulación son los más usados por la industria de manufactura, caracterizador por ser una mano robótica con más de 4 grados de libertad, dentro de sus funciones se puede encontrar desde el ensamblado hasta la manipulación del entorno donde se encuentre el robot.

Los sistemas móviles se consideran como una plataforma autónoma que puede desplazarse en el entorno, siendo usados en tareas agro-industriales, hasta tareas de navegación y planeación en entornos dinámicos.

Los sistemas de adquisición se utilizan para la transmisión y captura de información en el entorno, los cuales se encargan de garantizar el flujo de datos si se tiene una red en el sistema.

2.3. SISTEMAS ROBÓTICOS ASISTENCIALES

.

Los Agentes Robóticos Autónomos ARS han sido utilizados para ayudar a personas con algún tipo de discapacidad, tal como lo expresa Kawamura en su artículo titulado "Trends in Service Robots for the disabled and the elderly people" de 1994 [36], en donde establece un estado del arte en robots que ayudan a las personas con discapacidad, dando a conocer su diseño y una clasificación de estos, siendo retomado este tema por H. Qian en un capítulo de libro titulado "The State of the Art in Service Robotic System design" en el 2015 [60], donde re-clasifica a los ARS en tres categorías como se muestra a continuación:

1. **Sistemas estacionarios robóticos de servicio:** Se caracterizan por ser un sistema estático, normalmente se encuentran ubicados en un puesto de trabajo los cuales ejecutan tareas repetitivas. Son la base del desarrollo de la robótica, no solo a nivel de servicio, sino además a nivel industrial, tal como lo muestra la empresa ST robotics con su brazo robótico Fig. **2-4**.

2. **Sistemas móviles robóticos de servicio con accesorios:**El brazo y el actuador se instalan en una silla de ruedas, siendo un sistema dependiente en su totalidad por el conductor, un ejemplo es la silla de Stephen Hawking (Fig. **2-5**).



Figura 2-4.: Brazo robótico de ST Robotics
Fuente: `www.strobotics.com`



Figura 2-5.: Silla usada por Stephen Hawking
Fuente: `www.cnet.com`

3. **Sistemas de casa robóticos de servicio móviles:** Este sistema robótico es el que posee mayor movilidad y posee autonomía en su funcionamiento, además, aunque no dependan de una persona, deben tener la capacidad de interactuar con estas, un ejemplo es el robot asistente de limpieza de Irobot (Fig. **2-6**), el cual ejecuta tareas de limpieza en entornos dinámicos y estocásticos.

   Los ARS no solo se han usado para ayudar a pacientes con discapacidades físicas, estos también se han usado para ayudar a guiar a personas en entornos cerrados, como el robot RHINO (Fig. **2-7**) [75], el cual guía a las personas dentro de un museo, haciendo un proceso con un único agente, aumentando el tiempo para procesar el ambiente.



Figura 2-6.: Aspiradora robot de Irobot
Fuente: `www.irobot.com`



Figura 2-7.: Robot guía RHINO
Fuente: [75]

2.4. SISTEMAS MULTI-AGENTE (MAS)

Es un sistema auto-organizado compuesto por múltiples agentes que interactúan entre ellos para resolver problemas en un mismo entorno. Los agentes que componen el sistema pueden representados a nivel físico o en software, siendo estos al mismo tiempo clasificados como agentes pasivos y activos, siendo los agentes pasivos los encargados de representar obstáculos o unidades que no tienen un objetivo claro en el sistema; por otro lado, los agentes activos interactúan con el entorno para resolver objetivos, teniendo reglas simples en su programación o en su comportamiento (Fig. **2-7**), o reglas de aprendizaje (Fig. **2-8**), que le brinda al agente un grado de razonamiento para resolver problemas que no estaban previstos inicialmente y que le estén evitando cumplir los objetivos iniciales.
Los sistemas MAS se caracterizan por resolver tareas que son imposibles o tardan mucho

tiempo en ser resueltas por tan solo un agente, aumentando la autonomía, la tolerancia a fallos y el consumo de energía en el sistema.

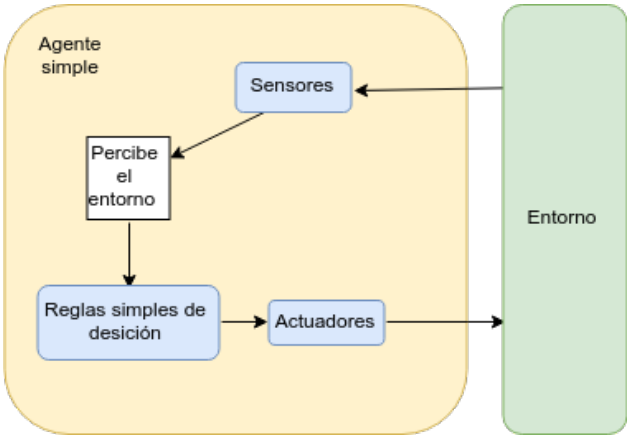


Figura 2-8.: Modelo de un agente con reglas simples
Fuente: Propia

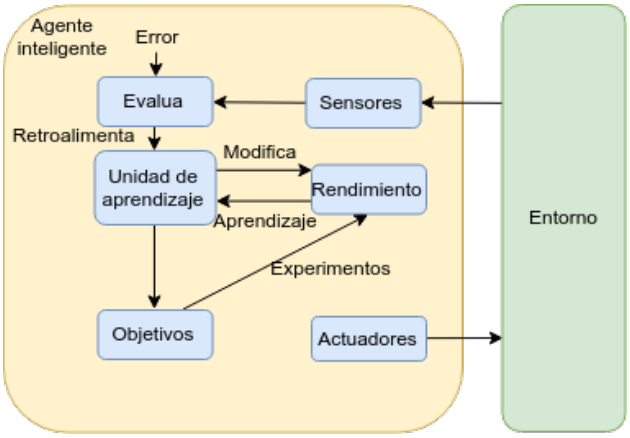


Figura 2-9.: Modelo de un agente con reglas de aprendizaje
Fuente: Propia

2.5. SISTEMAS MULTI-AGENTE ROBÓTICOS (MARS)

Son sistemas compuestos por robots autónomos que conforman una misma red, para interactúan en grupo cumpliendo un objetivo en común. Estos sistemas requieren de cooperación para poder moverse en entornos dinámicos y complejos sin control, en donde, la capacidad

de entender e interpretar el entorno es su prioridad [32], en la Fig. **2-10** se aprecia el diseño de un agente básico en MARS.

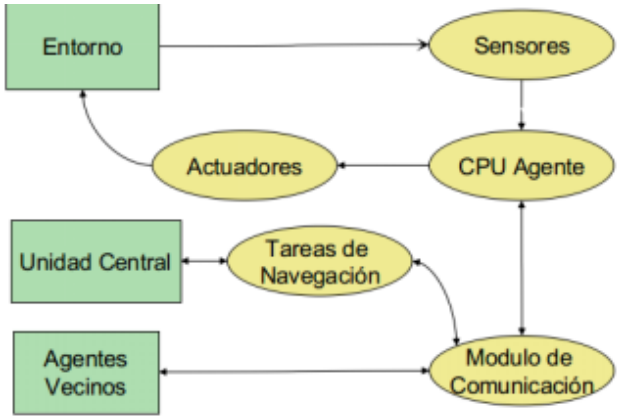


Figura 2-10.: Modelo básico agente robótico
Fuente: [32]

Estos sistemas robóticos deben tener arquitecturas concurrentes y distribuidas, con el fin de aprovechar el procesamiento en paralelo reduciendo el tiempo de cómputo, lo cual, emerge en una alta eficiencia orientada al manejo de los recursos, procesando tareas en tiempo real. Por otro lado los agentes del sistema deben ser modulares, tanto a nivel de software y de hardware, presentando componentes de alta cohesión y baja dependencia generando una alta robustez y alta tolerancia a los fallos al poder no tener una dependencia absoluta en las unidades que lo componen, y siendo el peor de los casos en que una unidad principal falle, se puede detectar ese fallo de manera sencilla y al mismo tiempo de solucionarlo al intercambiar el componente [75, 7, 24].

2.6. REDES DE SENSORES INALÁMBRICAS

Las redes de sensores inalámbricas pueden ser vistas como un sistema distribuido compuesto por pequeños sistemas embebidos, cada uno de estos se compone por una unidad de procesamiento con una interfaz de comunicación inalámbricas, y una o más unidades de sensores [50] tal como se muestra en la Fig. 11. Los nodos que componen la red, permiten la captura de la información en entornos de riesgo, enviando los datos a un servidor para ser procesados. La flexibilidad de las WSN permite ser aplicada en tareas de monitorio del entorno en ambientes peligrosos o de alto riesgo para las personas; en la robótica, permite la distribución de la información para realizar SLAM; en el área de la salud, permite el monitoreo de pacientes en

hospitales de forma remota; y en sistemas de seguridad, permite la detección de disturbios en el entorno en que son instaladas [27].

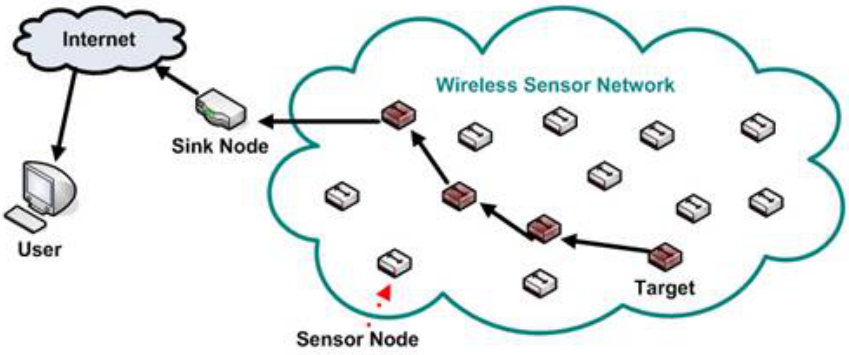


Figura 2-11.: Esquema Red de Sensores Inalámbricas
Fuente: `monet.postech.ac.kr/`

Para el procesamiento de datos se requieren tres etapas: Primero el envió de datos entre los nodos a una unidad de almacenamiento debe ser con una frecuencia que tanto el nodo como la unidad principal pueda soportar. Segundo la transmisión y recepción de datos debe ser controlada para evitar datos inválidos o perdidas de paquetes. Tercero la unidad principal encarga de subir los datos al servidor debe ser capaz de controlar el flujo de la información en los nodos [58, 59].

2.7. ALGORITMOS DE CONSENSO

Los algoritmos de consenso permiten que una colección de máquinas puedan trabajar como un grupo coherente con una alta tolerabilidad a los fallos incluyendo si uno o más de las máquinas que lo componen llegaran a colapsar, por esto los algoritmos de consenso juegan un papel muy importante para la construcción de sistemas de gran escala confiables [54].
La estructura de los algoritmos de consenso se desarrolla en el contexto de máquinas de estado replicables, supongamos el caso de un sistema cuyos clientes envían comandos y reciben respuestas, pueden ser modelados como una máquina de estados, en donde al ejecutar un comando en un respectivo estado genera a la salida un nuevo estado [40]. En este enfoque las máquinas de estado son representadas como copias idénticas de los estados en la colección de servidores (ver Fig. **2-12**), de las cuales pueden seguir ejecutándose si algún servidor dentro de la colección colapsa.

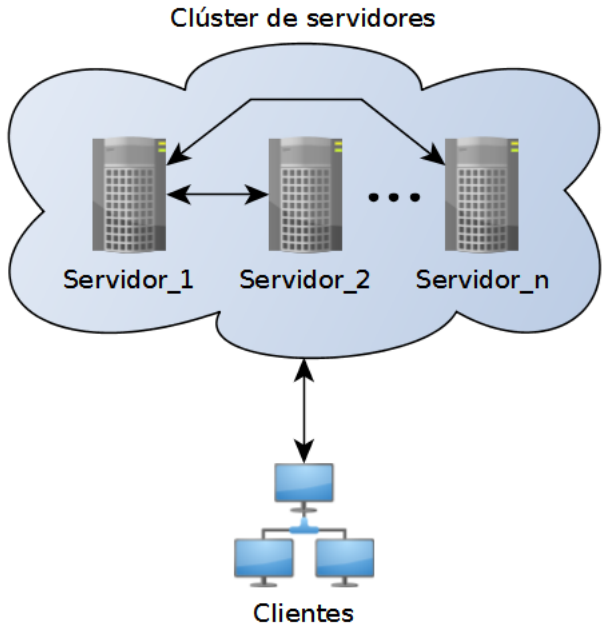


Figura 2-12.: Clúster de servidores usados junto con los clientes
Fuente: Propia

Estas máquinas de estado son implementadas usando réplicas de los estados de los comandos enviados por los clientes como se muestra en la Fig. **2-13**.

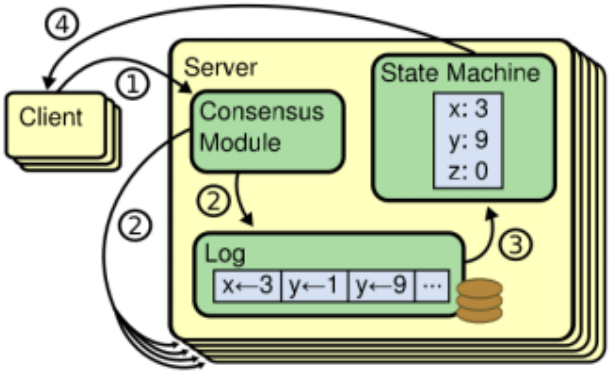


Figura 2-13.: Arquitectura de la máquina de estados replicada
Fuente: [54]

Cada uno de los servidores almacena la lista de los comandos, siendo ejecutados de manera ordenada por la máquina de estados en todos los servidores de manera simultánea eliminando la incertidumbre. Las propiedades fundamentales de los algoritmos de consenso se citan a continuación [54]:

1. Garantizan la seguridad del sistema, estos nunca retornan un valor incorrecto, incluyendo retardos, particiones, pérdida de paquetes, duplicación o re-ordenamiento, solamente si se asegura que el sistema no tenga condiciones arbitrarias. [1, 80].

2. Son totalmente funcionales si la mayoría de los servidores están activos y puedan comunicarse entre sí y a la vez con los clientes.

3. Estos no dependen del tiempo para garantizar la lista de comandos en los servidores.

4. En modo normal, un comando puede ser completado tan pronto como la mayoría de los clúster ha respondido a las acciones de los clientes.

Parte II.

DESARROLLO

3. CAPÍTULO 3: MODELO DE INTEGRACIÓN BASADO EN EL CONCEPTO DE WSN

3.1. INTRODUCCIÓN

Las WSN se caracterizan por su flexibilidad y modularidad, permitiendo ser utilizadas en tareas de navegación en la robótica, tales como el mapeo y localización del entorno. Estas tareas se realizan mediante la lectura del parámetro de indicador de fuerza de la señal recibida (RSSI) aplicando técnicas de fusión de información entre los diferentes agentes, a partir del procesamiento de la información por una unidad central [85, 10, 9, 65].

El problema de tener dependencia sobre una unidad central para el funcionamiento del sistema, radica en la necesidad del permanente funcionamiento de esta, en caso contrario el sistema colapsaría. Por otro lado, la unidad central necesita que el sistema siempre mantenga una comunicación, de esta manera, todos los nodos deben comunicarse hacia la unidad central, necesitando de rutas indirectas de comunicación si la unidad central se encuentra fuera del radio de cobertura, aumentando el consumo de energía y la complejidad computacional al implementar algoritmos de enrutamientos sobre los nodos [18].

En este capitulo, se presenta un modelo integrando los MARS con la flexibilidad de las WSN, evitando la dependencia de una unidad central, orientado a la autonomía del agente robótico en tareas de navegación descentralizada.

3.2. APLICACIÓN DE LAS WSN EN LOS SISTEMAS MULTI-AGENTE

El principal objetivo de los MARS es darle autonomía a cada uno de los agentes que integra el sistema (Fig. **3-1**), esto es, cada agente debe tener la propiedad de tener la capacidad de procesar los datos de manera distribuida y descentralizada, también, deben tener un sistema operativo que garantice la administración de tareas de procesamiento, permitiendo

la operación del sistema en tiempo real, sin olvidar las propiedades de tener una arquitectura híbrida, la generación de modelos, la capacidad de aprendizaje y la modularidad de los agentes que componen el sistema [75, 56, 86].

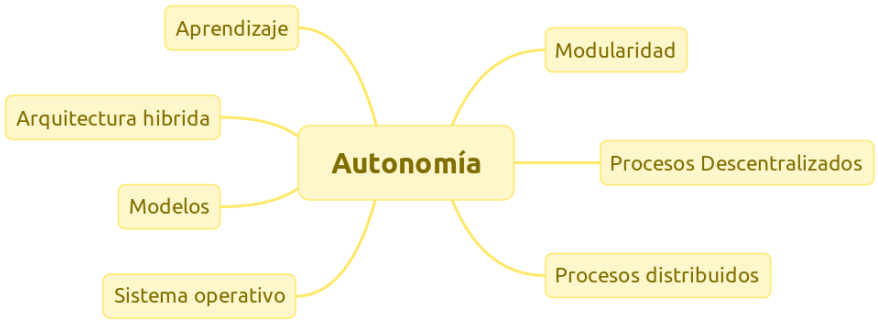


Figura 3-1.: Mapa mental de autonomia en agentes robóticos
Fuente: Propia

En las WSN también se ha usado el concepto de autonomía, siendo este orientado a trabajar con Sistemas Multi-Agentes (MAS), describiendo cada nodo de sensado como una tecnología capaz de ser ubicada en cualquier entorno, percibiendo las características de este (dependiendo de los sensores), y basado en la dinámica del entorno se ejecutan planes de acción pre programados para interactuar directamente por medio de actuadores, o se envían señales a la unidad central para la toma de decisiones [48]. Estos planes de acción están orientados a la tareas de auto-mantenimiento, auto-protección, auto-optimización y auto-configuración, con el fin de reducir recursos de computo y consumo de energía, siendo un excelente candidato para ser empleadas de forma directa en agentes móviles como los agentes robóticos (AR) orientados a tareas de navegación [78].

Lo anterior permite ampliar el estudio de las WSN en temas de movilidad, logrando simplificar las tareas de procesamiento de los nodos sensores y clasificando el tipo de movilidad en estos. Para aumentar la autonomía, reducir el tiempo de procesamiento y aumentar el tiempo de vida de las baterías en los nodos, es necesario conocer el entorno donde se implementaran los nodos de la red, con el fin de clasificar los nodos acorde a las tareas que van a desempeñar. Esta categorización depende del tipo de movilidad de los nodos en el entorno, siendo clasificados como nodos con movilidad débil o fuerte. La movilidad débil describe a nodos sensores que cambian sus características por el nivel de batería, y los nodos de movilidad fuerte son los nodos que tienen la capacidad de moverse por cuenta propia como en el caso de la robótica [71].

Dado que los sensores de movilidad fuerte describen a los AR, estos se han usado para realizar técnicas de localización, mapeo y planeación en entornos cerrados y abiertos, en el caso de Caballero *et al.*, [8] propone un framework para la localización de nodos inalámbricos por medio de un AR usando un modelo probabilístico bayesiano en la clasificación de pesos en los filtros de partículas, siendo el parámetro RSSI el encargado de brindar los pesos en el filtro sin necesidad de realizar tareas de triangulación para el calculo de distancia entre los sensores y el AR, esto permite realizar una estimación en tres dimensiones de la localización de los agentes en el entorno, pero no logra generar un mapa.

Por otro lado, se han trabajado técnicas para localizar al AR en un entorno desconocido, ubicando nodos de sensores en el ambiente, de manera que brinden la información de la posición por medio de triangulación entre el AR y los diferentes nodos, para realizar tareas de navegación [22]. Usando este enfoque, se han realizado tareas de planificación en entornos desconocidos, posicionando al AR en un patrón en forma de grilla, logrando detectar obstáculos para llegar al punto objetivo [85]. El problema de esos enfoques es que requieren que con anterioridad se hayan instalado los sensores en el entorno, y para lograr una buena estimación del espacio, requieren una gran cantidad de nodos sensores en el entorno, además, en el sistema descrito no presentan un plan para solventar si un nodo estático pasa a hacer un nodo de movilidad débil.

Para eliminar la dependencia de nodos estáticos en los entornos, se han trabajado enfoques para posicionar nodos de sensores por medio de un AR para monitorear zonas de desastres. Schwager *et al.*, usa un vehículo aéreo sin nombre (UAV) para posicionar sensores en un entorno sin infraestructura de comunicación, este sistema se encarga de crear una WSN entre los nodos estáticos posicionados por el UAV, nodos en movimiento y una estación de monitoreo para el procesamiento de los datos, los autores expresan que al posicionar los nodos, encontraron problemas de comunicación con el nodo instalado por la posición de la antena, además, no se trabajan técnicas para estudiar la localización del nodos sobre el entorno. Cheng *et al.* [14] propone un MARS para trabajar en entornos de emergencia cooperativa para realizar SLAM y detectar personas en este entorno por medio de nodos de sensores PIR inalámbricos. Los AR que componen el sistema, se encargan de trabajar por medio del enfoque de exploración por roles [72, 13] de exploradores y transmisores, en donde, los agentes encargados de la transmisión entre la unidad central y los exploradores, evalúan el nivel de RSSI entre los agentes para posicionar nodos de sensores y que los exploradores se comuniquen entre sí y la unidad central para realizar el SLAM de manera cooperativa (CSLAM) [3, 74]. Este modelo logra resultados en simulación que demuestran la autonomía del sistema para el posicionamiento de los nodos inalámbricos. Sin embargo, la autonomía del sistema se ve comprometida al depender de una unidad central.

Por otro lado, se han propuesto múltiples estrategias de control descentralizado, eliminando la dependencia de una unidad central en los MARS, al solo compartir la información con el vecino más cercano en el sistema. Para la distribución de sensores, Schwager et. al. [37] asigna a cada agente dentro del MARS un sensor para que estos lo instalen en un área usando algoritmos de consenso para determinar sus posiciones finales, limitando su estudio en un ambiente convexo y no en un entorno real.

En tareas de cobertura, mapeo, vigilancia y corrección de formaciones, las estrategias de control han mostrado la descentralización del sistema en entornos no convexos, pero no estiman la interacción con otros agentes heterogéneos al sistema, es decir, no permiten la incorporación de diferentes agentes [4, 28, 68, 66]. Otros enfoques muestran la interacción con diferentes tipos de agentes robóticos, siendo sistemas heterogéneos al tener Unamed Aereal Vehicles (UAVs) y Unamed Ground Vehicles (UGVs) [72], donde la descentralización se hace con un sistema de control con comunicación permanente entre las unidades terrestres, planteando un esquema de comunicación unidireccional entre las unidades terrestres y las aéreas, pero no permite el intercambio de información entre los diferentes grupos de agentes para aumentar la tolerancia a fallos.

3.3. MODELO DEL SISTEMA

El modelo para representar un agente robótico en MARS haciendo uso de WSN en navegación se muestra en el diagrama de casos de uso en la Fig. **3-2**, donde el entorno entrega una colección de datos capturados por los diferentes sensores del agente robótico para enviarlos a una unidad central encargada de la toma de acciones. La unidad central al procesar la información, envía las acciones pertinentes a los agentes y estos las ejecutan. Cabe resaltar que las acciones ejecutadas por los agentes robóticos dependen de las características espaciales del entorno, es decir, la elevación, tipo de terreno, etc. las cuales son parcialmente analizadas por los sensores.

El modelo de la Fig. **3-2** presenta un gran inconveniente, si el sistema central falla, todos los agentes involucrados en el MARS se pueden perder, dado a que siempre deben comunicarse con la unidad central. De otra manera si la unidad central es robusta y nunca fallara, siempre debe interactuar como mediador entre los agentes que no tienen comunicación directa, aumentando el tiempo para la toma de decisiones, afectando de forma directa el consumo de energía y la eficiencia computacional del agente robótico quitándole autonomía al sistema. Por tal motivo se presenta el diagrama de secuencia mostrado en la Fig. **3-3** para el diseño del agente robótico, modelando el flujo del sistema, desde la captura de datos, la ma-

nipulación de los actuadores a la comunicación de los agentes dentro del MARS de manera descentralizada.

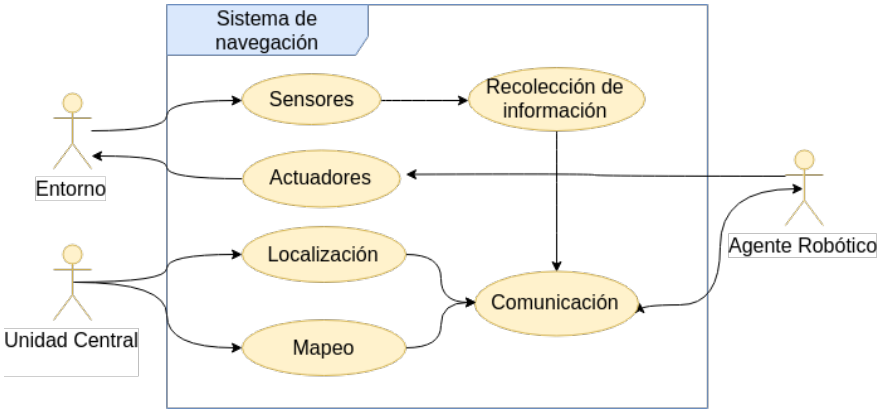


Figura 3-2.: Diagrama de casos de uso del diseño básico del agente en WSN centralizado
Fuente: Propia

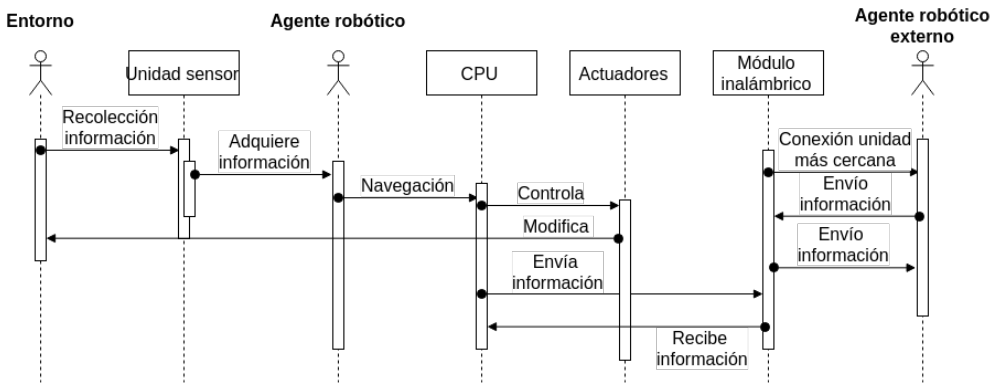


Figura 3-3.: Diagrama de secuencia para el diseño del agente robótico descentralizado con WSN

Fuente: Propia

El diagrama en la Fig. **3-3**, toma como referente la tecnología de WSN para integrarla con los MARS, generando un modelo denominado DWMARS; cada agente robótico (nodo) que compone el MARS tiene una unidad con sensores, una unidad de procesamiento, un modulo de comunicaciones inalámbrico y una unidad de actuadores que normalmente esta compuesta de unidades de control y motores. Cada nodo evalúa el entorno por medio de los sensores y de la información enviada por los demás agentes mediante el modulo de comunicaciones inalámbrico, siendo esta analizada por la unidad de procesamiento, realizando tareas de SLAM [1, 8, 19, 22, 23, 27, 20] y planeación de manera simultanea, sin necesidad de esperar

respuesta de una unidad central, reduciendo el consumo de energía al realizar la comunicación nodo a nodo basada por la RSSI entre estos.

3.4. MODELO DE COMUNICACIÓN BASE ENTRE LOS AGENTES

El modelo descentralizado de la Fig. **3-3** genera la comunicación con el nodo mas cercano a nivel de potencia, basándose en el parámetro de indicador de fuerza de la señal recibida (RSSI).

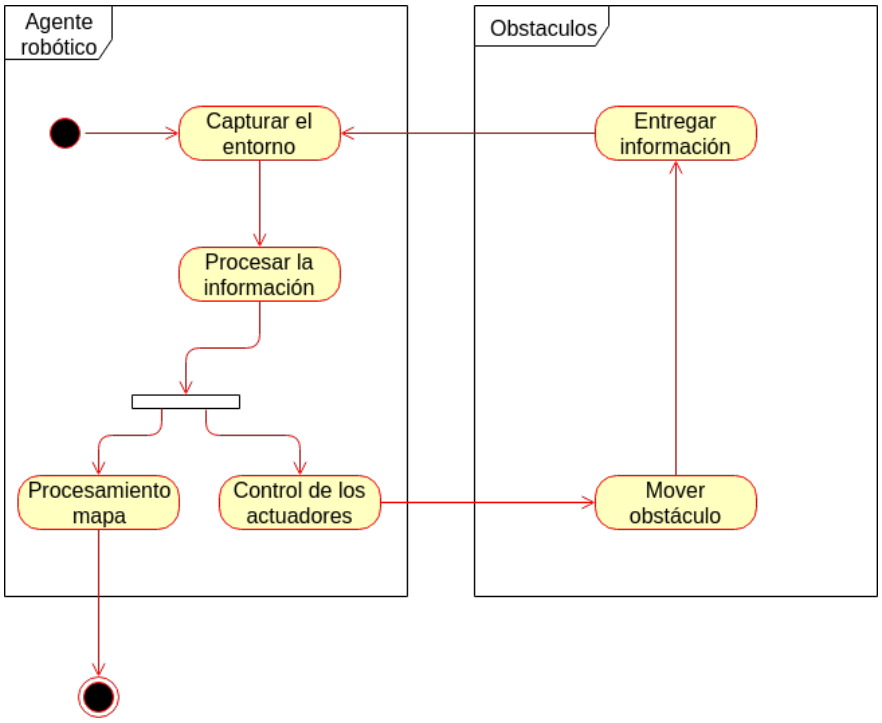


Figura 3-4.: Diagrama de trabajo comunicación y procesamiento del agente robótico
Fuente: Propia

La comunicación se caracteriza por parametrizar cada nodo con un nombre especifico, que debe ser incluido en una lista y esta debe ser compartida entre todos los nodos del sistema, por tanto, se tiene este único requisito para pertenecer al sistema obviando el tipo de sensores que maneje el agente robótico. Las tareas de navegación son realizada por los nodos con el vecino más cercano(de tener un enlace) basándose en diferentes técnicas, tal y como se expresa en [41, 12, 79, 64], evitando la necesidad de comunicarse con una unidad central.

Este funcionamiento es representado en el diagrama de flujo de trabajo de la Fig. **3-4**.
El anterior diagrama de trabajo se representa en el algoritmo 1 (ver **anexo 1**).

Algoritmo 1 Modelo de comunicación entre n agentes para compartir un mapa

```
1:  Procedimiento: COMUNICACIÓN
2:      listaAgentes ← agentesCercanos
3:      mejorRSSI ← −∞
4:      agente ← −1
5:      for all nodo ∈ listaAgentes do
6:          if nodo.RSSI > mejorRSSI then
7:              mejorRSSI ← nodo.RSSI
8:              agente ← nodo.id
9:      conectar(agente)
10:     verificaInformacion :
11:     timeOut ← rand(10ms → 100ms)
12:     if idTransferenciaPrevia ≠ agente.id then
13:         if llaveUltimaTrasnferencia ≠ agente.llave then
14:             fusionarMapa(agente.mapa)
15:     else
16:         fusionarMapa(agente.mapa)
17:     estimarUbicacion(mapa)
18:     if error > ϵ then
19:         goto → verificaInformacion
```

El anterior algoritmo puede ser usado en tres diferentes escenarios, a manera de ejemplo se dejan tres agentes distribuidos en un entorno de manera aleatoria. El primer escenario es mostrado en la Fig. 3.4, en donde se puede apreciar que ningún agente esta lo suficientemente cerca del otro para establecer comunicación por medio de su RSSI, en este caso cada agente se encarga de realizar las tareas de SLAM de manera individual hasta encontrar otro agente lo suficientemente cerca. EL segundo escenario es mostrado en la figura Fig. 3.4, donde se encuentran nodos lo suficientemente cerca para establecer la comunicación, siendo en este caso el *agente a* y el *agente c*. En el tercer escenario se encuentra un muro a manera de obstáculo; en este escenario, los tres agentes deben buscar el agente más cercano para establecer la comunicación y empezar la tarea de localización y mapeo; en este escenario se puede observar que el *agente a* esta mas cerca al *agente b* en parámetros de distancia, pero estos se encuentran separados por un muro que reduce de forma significativa el RSSI entre estos, mientras el *agente c* se encuentra en linea de vista con el *agente a* teniendo un mayor

RSSI en comparación con el nodo *agente b*, permitiendo la comunicación como se muestra en la Fig. 3.4.

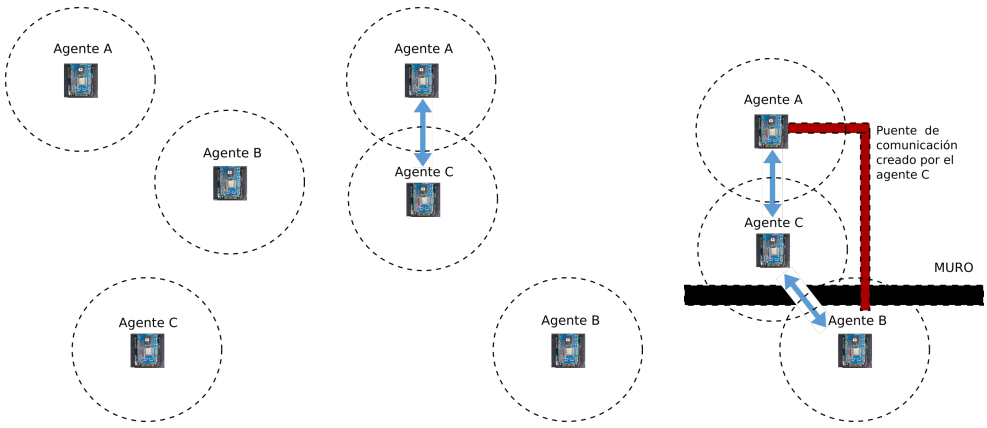


Figura 3-5.: (a) Agentes distribuidos sin conexión. (b) Conexión entre los agentes basándose en el RSSI (c) Conexión entre los agentes basándose en el RSSI con un obstáculo

En este modelo se pueden aplicar técnicas de filtros de partículas, permitiendo realizar la localización de los agentes robóticos a nivel individual mientras que se trabaja el mapeo del entorno. El algoritmo básico de filtros de partículas es presentado a continuación donde se definen las funciones de medición y de movimiento:

```
def sense(likelihood, sensor):
    temporal= [[] for i in range(len(likelihood))]
    for i in range(len(likelihood)):
        for j in range(len(likelihood[0])):
            hit = (sensor == environment[i][j])
            temporal[i].append(likelihood[i][j]
            * (hit * pSense + (1-hit) * (1-pSense)))
    s = 0.0
    for i in temporal:
        s+=sum(i)
    if (s>0.0):
        for i in range(len(likelihood)):
            for j in range(len(likelihood[0])):
                temporal[i][j] /= s
    else: print "Agent lost"
```

```
def move(likelihood, movY, movX,pMove):
    ii = movX%len(likelihood[0])
    jj = movY%len(likelihood)
    pNMove = 1-pMove
    temporal = [(t[-ii:]+t[0:-ii])
    for t in (likelihood[-jj:]+
    likelihood[0:-jj])]
    for x in range(len(likelihood)):
        for y in range(len(likelihood[0])):
            temporal[x][y] = temporal[x][y]*
            pMove+likelihood[x][y]*pNMove
    return temporal
```

3.5. EVALUACIÓN DEL MODELO

El modelo es aplicado en dos sistemas. El primer sistema se caracteriza por estar compuesto solamente por un agente robótico y múltiples agentes inalámbricos, los cuales son usados para localizar al agente en un entorno desconocido y generar el mapa de dicho entorno de manera descentralizada. El segundo sistema esta compuesto por múltiples agentes robóticos y agentes heterogéneos, que comparten la información con el fin de lograr la misma tarea propuesta por el sistema multi-agente del primer sistema.
Los resultados del primer sistema demuestran una mejora significativa frente a los sistemas centralizados, al recuperarse de una falla de comunicación; mientras que frente a otros sistemas descentralizados, muestra el potencial para reducir el consumo de los recursos. Estos resultados son evidenciados en la tabla 3.5, siendo explicado el proceso de validación el capitulo 4.

Descentralizado		**Centralizado**	
WA usados	tiempo $(mm:ss)$	WA usados	tiempo $(mm:ss)$
9	22 : 20	16	16 : 43

Tabla 3-1.: Resultados del modelo aplicado en un sistema MAS frente a un sistema centralizado.

El segundo sistema permite validar la interconexión de múltiples agentes a un sistema MARS.

Demostrando la reducción de datos enviados entre las unidades que componen el sistema, y al mismo tiempo ser tolerante a los fallos al no depender de una unidad central. Estos resultados son evidenciados en la tabla 3.5, siendo explicado el proceso de validación el capitulo 5.

Tipo	Robot IP	Distancia (m)	Tiempo (s)	Bytes enviados
Centralizado	192.168.1.2	30.5	472.46	–
	192.168.1.85	23.85	447.69	–
	192.168.1.170	33.3	484.35	–
	192.168.1.86	–	–	–
Descentralizado	192.168.1.2	18.63	297.14	48
	192.168.1.85	29.20	523.93	81
	192.168.1.170	6.67	109.70	69

Tabla 3-2.: Resultados del modelo aplicado en un sistema MARS frente a un sistema centralizado.

4. CAPÍTULO 4: APLICACIÓN DEL MODELO EN SISTEMAS ROBÓTICOS MULTI-AGENTE

4.1. INTRODUCCIÓN

En el capítulo anterior se presento un modelo de integración que permite la comunicación entre AR de manera descentralizada para realizar tareas de navegación. Dentro de estas, se encuentran procesos de planeación, pero para esto se requiere que con anterioridad, el AR tenga el mapa del entorno y el lugar en el cual se encuentra ubicado, siendo un problema cuando el agente robótico se ubique en un nuevo entorno. Para lograr superar este problema el AR debe realizar la tarea de localización y mapeo simultaneo (SLAM), logrando ubicar al agente en el entorno mientras genera el mapa sea de manera geométrica o topológica.
Uno de los grandes problemas al realizar SLAM es la cantidad de memoria que requiere el RA para almacenar los detalles del mapa del entorno, además, para capturar detalles del entorno se requiere una unidad robusta de procesamiento para la representación de datos, que refleja un RA que aumenta el consumo de energía, los costos de fabricación y también dependa de una unidad externa para el procesamiento de los datos.
En este capitulo se presenta el diseño de un sistema para hacer SLAM de manera descentralizada, usando el modelo propuesto en el capitulo anterior, usando varios nodos que son representados por agentes inalámbricos, que permiten el almacenamiento y la distribución del mapa generado por un AR.

4.2. ARQUITECTURA DEL SISTEMA

La navegación en un entorno desconocido es una de las tareas más difíciles que tiene un agente robótico (RA), estas sacrifican la autonomía al no conocer la posición, el mapa ni las posibles rutas que se podrían tomar para cumplir el objetivo propuesto en el RA. En la actualidad existen técnicas para la localización y el procesamiento del mapa (SLAM), las cuales permiten que un agente o un grupo de agentes, capturen información del entorno para

ubicarse e ir generando el mapa por medio de sensores.

Uno de los sensores más usados para el SLAM son los de telemetría láser (LIDAR), estos se caracterizan por medir distancias al enviar un pulso de luz infrarroja o láser, que al impactar sobre un objeto se refleja la onda que es capturada por un escaner para su procesamiento [21, 2, 43]. Además de los sensores LIDAR, también se usan cámaras monoculares o estereoscopicas que capturen la información del entorno, las imágenes son procesadas para generar el mapa en tres dimensiones [67].

Sin embargo, tanto los sensores LIDAR y las cámaras, requieren una gran cantidad de memoria para el procesamiento y representación del mapa, al estimar la posición del RA por medio de técnicas de predicción y de optimización como filtros bayesianos, filtros de partículas y filtros de Kalman con distribuciones normales [17, 16, 19].

Para la reducción del procesamiento de información para la localización del RA en un entorno desconocido, se hace uso de la tecnología de redes de sensores inalámbricos (WSN), en especial al patrón de radiación de potencia (RSSI) producido por los nodos inalámbricos que conforman la red. Este parámetro se usa para triangular al RA en el entorno, requiriendo que al menos la señal de tres nodos detecten al RA en la red para estimar la posición su posición por medio de una unidad de procesamiento externa. Además de obtener la posición, la ubicación a priori de los sensores permite que el RA genere un mapa topológico del entorno, al caracterizar cada nodo de la red como un vértice del grafo del mapa, reduciendo la complejidad y la cantidad de memoria para el procesamiento de la localización la generación del mapa [85].

Sin embargo, al requerir que los nodos de la WSN sean ubicados a priori y que el procesamiento sea realizado por una unidad externa reduce la autonomía del sistema. Además, se requiere que cada nodo sea programado con la ubicación especifica en el mapa de forma uniforme, lo cual, dificulta la instalación en entornos de difícil acceso.

A continuación se presenta el diseño de un sistema para el desarrollo de SLAM descentralizado, haciendo uso de un RA que instale agentes estáticos inalámbricos (WA) en el entorno, los cuales construyen un sistema multi-agente (MAS) basado en el principio mostrado en el capítulo 3 para la comunicación y generación de mapas topológicos. Para esto, se propone un RA diferencial equipado con sensores de distancia para la detección de obstáculos, un compás digital que permita orientar al RA, y encoders en cada uno de los motores para la estimación del espacio recorrido en el entorno, el cual, se encarga de programar e instalar los nodos en el entorno. Para iniciar con la localización, el RA inicia con la creación de un grafo, en el cual, cada nodo va a representar una posición (x, y) en $\mathbb{R}^2$ y las conexiones la continuidad en los trazos para el mapa topológico. El grafo va agregando nuevos nodos a medida que el RA detecta obstáculos o cruces entre los trayectos recorridos, siendo sus

valores los procesador por la odometría[3] en los motores y por la orientación suministrada por el compás digital. Por último, la instalación de los WA se realiza cuando el RA examina el entorno y no detecta en el rango basado en el RSSI a otro WA, durante este proceso el grafo es programado en el WA y es constantemente actualizado si se encuentra en el rango de cobertura, esto permite que si el RA sufriera una avería, el mapa pueda ser cargado por un nuevo RA.

4.2.1. Descripción de los agentes

El sistema multi-agente se compone de un RA y múltiples WA. El RA es el encargado de generar el mapa on-line del entorno mientras que selecciona los puntos críticos para la instalación de los WA. Durante el proceso de instalación de los WA, el RA se encarga de programar el nodo con el mapa y la posición estimada acorde al entorno.
En este trabajo de investigación, se usa un RA compuesto por cinco sensores de distancia para la detección de obstáculos en el entorno, permitiendo la evasión y al mismo tiempo la selección de rutas posibles en el entorno. Además esta compuesto por un compás digital encargado de orientar al RA, dos motores con encoders para la estimación de la distancia recorrida, una unidad de procesamiento que controla el movimiento y la generación del mapa, y una unidad inalámbrica para la comunicación y programación de los WA.

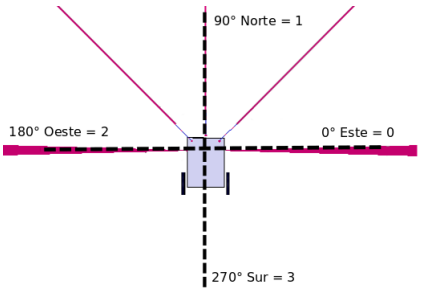


Figura 4-1.: Asignación de identificadores del AR respecto el plano cartesiano.
Fuente: Propia.

El sistema de localización toma como núcleo el compás digital y la odometría de los motores. El compás digital ubica la dirección del agente para incrementar la posición en x o en y. Para realizar el incremento de la distancia recorrida, se caracterizan las cuatro posiciones cardinales acorde al valor entregado por el sensor, siendo 0° grados el este, 90° grados norte, 180° grados oeste y 270° grados sur. Cada una de las posiciones cardinales son almacenadas

[3]Estimación del cambio de la posición con respecto al tiempo

en la memoria del AR y de las WA, mediante de un identificador entero del 0 al 4 como se puede ver en la Fig. **4-1**.
La posición recorrida es estimada a partir de los encoders en los motores. Sin embargo, la medida entregada por los encoders presenta errores por la diferencia del tamaño de las llantas y la resolución discreta que presenta por la lectura, estos errores se clasifican como errores sistemáticos. Para la reducción de estos errores, se implementa el modelo UMBmark [63], sobre el RA, teniendo como base el modelo cinemático descrito por la matriz de translación y rotación de la ecuación (4-1).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \vec{r} \\ \theta \end{bmatrix} + \begin{bmatrix} x \\ y \\ \phi \end{bmatrix} \tag{4-1}$$

Donde:
$\dot{x}$ = Nueva posición eje x.
$\dot{y}$ = Nueva posición eje y.
$\dot{\theta}$ = Nuevo angulo.
$\vec{r}$ = Vector velocidad del AR
θ = Angulo de rotación.
x = Posición actual con respecto al eje x.
y = Posición actual con respecto al eje y.
ϕ = Angulo actual.

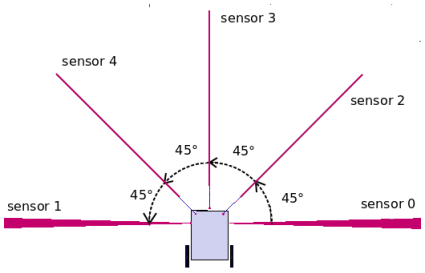


Figura 4-2.: Ubicación de los sensores de proximidad.
Fuente: Propia.

Con el fin de evitar colisiones y detectar las rutas que pueden ser recorridas, los sensores de proximidad son instalados con una separación 45° grados y están ubicados en la parte

delantera del chasis del RA. Los sensores con el identificador 2, 3 y 4, ubican los obstáculos que puedan obstruir el movimiento frontal, y los sensores con el identificador 0 y 1, detectan los trayectos o los obstáculos laterales Fig. **4-2**.

4.2.2. Puesta de los sensores e intercambio de la información

La unidad inalámbrica de RA se encarga directamente de comunicarse con los WA. Durante el proceso de comunicación se valida si existen nuevos nodos para agregar a las unidades, esto con el fin de compartir el mapa entre todos los WA que componen la red, además, se encarga de validad si existen WA en el rango de cobertura acorde al parámetro RSSI, si no existe un WA en el rango de cobertura, el RA se encarga de realizar la programación del nodo. La programación del nodo inicia asignándole el SSID acorde al nombre del RA y al consecutivo de instalación del mismo. Luego la información se almacena por medio de arreglos dinámicos para representar el mapa. El arreglo tiene cuatro posiciones como se muestra en la tabla **4-1**, la primera posición indica el numero del nodo del mapa topólogico, las posiciones dos y tres indican la posición del nodo en el entorno y la última posición es nodo con el cual se conecta.

Tabla 4-1.: Arreglo dinámico de memoria para el agente inalámbrico.

Historial WA			
ID_{node}	pos_x	pos_y	$UltimoID_{node}$

Para que el RA pueda realizar la programación de un nodo, es necesario generar un historial en su memoria. El historial es almacenado como un arreglo dinámico, siendo las primeras 4 posiciones las mismas que los nodos WA, agregando una posición para almacenar las rutas que no se han recorrido en los nodos tabla **4-2**, esta posición almacena una cadena de números enteros que identifican la posición cardinal pendiente, las cuales están separados por un punto.

Tabla 4-2.: Arreglo dinámico de memoria para el agente robótico.

Historial RA				
ID_{node}	pos_x	pos_y	$UltimoID_{node}$	Trayectos pendientes

El RA actualiza su historial acorde a los valores entregados por los sensores. Si los sensores delanteros detectan un obstáculo que puede ocasionar colisión, o los sensores laterales detec-

tan un espacio por el cual el RA puede navegar, hacen que se actualice de forma inmediata el historial. En la Fig. **4-3**a, el RA se encuentra en la posición inicial, en ese instante se crea el primer identificador con las condiciones iniciales, estás son la posición $(0, 0)$ en $\mathbb{R}^2$, además detecta en las posiciones pendientes la posición 1, la cual implica que puede avanzar al norte respecto al valor entregado por el compás digital. El agente valida el historial para detectar las rutas por las cuales puede avanzar, este actualiza el historial y avanza. EL RA avanza hasta que el sensor de distancia detecta una colisión Fig. **4-3**b, esto ocasiona nuevamente que se actualice el historial con el valor de la posición procesado por la odometría; además agrega las posiciones 0 y 2 como rutas posibles para avanzar al arreglo. El agente nuevamente revisa su historial, selecciona de forma aleatoria una de las dos posiciones y ejecuta la acción de movimiento Fig. **4-3**c, retirando de la lista pendiente la posición (tabla **4-3**). Este proceso se repite hasta que el agente robótico detecte una colisión frontal y no se tengan trayectos pendientes en el historial del RA. En algoritmo 2 (ver: **anexo 2**) se muestra el modelo de actualización del historial del RA.

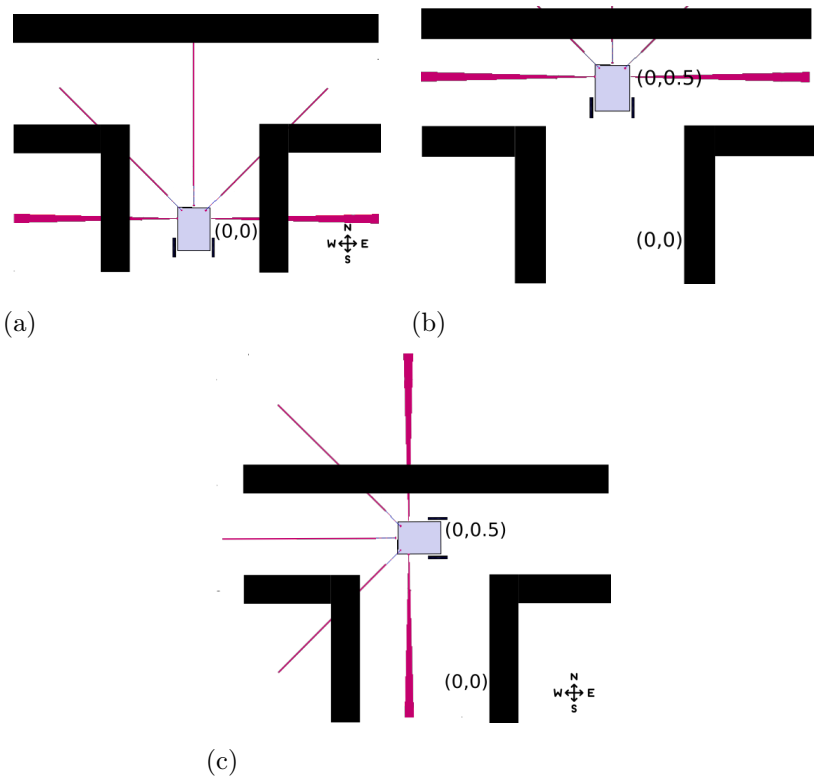


Figura 4-3.: Actualización arreglo del AR (**a**) posición inicial; (**b**) agregando un nuevo nodo; (**c**) actualizando caminos pendientes.

Tabla 4-3.: Actualización del arreglo dinámico del AR

Historial RA				
ID_{node}	pos_x	pos_y	$UltimoID_{node}$	Trayectos pendientes
0	0	0	0	-
1	0	0,5	0	0

La comunicación entre los agentes se realiza de manera descentralizada, donde los WA pueden desempeñar el rol de Access Point o de cliente, mientras que el RA se va a desempeñar como cliente. Este intercambio de información toma el modelo multi-agente de Xiong et al. [81], con la excepción que los WA siempre van a estar en cobertura de comunicación, y son los encargados de compartir y generar el mapa on-line entre todos los nodos se la red, mientras que el RA, solo va a programar al WA que tenga mayor RSSI, reduciendo los errores de pérdida de paquetes y la necesidad de siempre estar transmitiendo datos a una unidad central, aumentando la autonomía de la batería en los agentes del sistema.

Cuando un RA entra a un nuevo entorno o WA es instalado, este toma inmediatamente su configuración como un cliente para buscar otros WA en cobertura. Si encuentra a otro WA, se inicia un proceso de intercambio de información, siendo el agente que contenga más información almacenada el que envía la información. En el caso que el intercambio de información sea entre un RA y un WA, el RA almacena el SSID y la cantidad de datos intercambiados para evitar nuevamente una conexión. En el caso que el RA almacene un nuevo nodo del mapa, y el anterior WA aún se encuentre en cobertura, se vuelve a establecer la conexión para la transferencia de la información.

Por otro lado, si el intercambio de información es entre WA's, la unidad transmisora al terminar el proceso, cambia su configuración a access point, mientras que la otra unidad queda configurada como cliente, repitiendo el proceso nuevamente con los WA cercanos. Para evitar un lazo indefinido se debe evitar establecer comunicación con la anterior unidad transmisora si esta no tiene nueva información. Para esto, la unidad receptora almacena el SSID del WA que le transmite la información, junto con la cantidad de nodos transmitidos. Si la Unidad cliente no se conecta a otra unidad en un intervalo de 5 segundos, esta cambia su configuración a access point.

En la Fig. **4-4** se muestran dos WA que van a iniciar el proceso de intercambio de información. La unidad WA_R1_2 inicialmente se encuentra instalada en modo cliente y no contiene ningún SSID almacenado, mientras que la unidad WA_R1_1 se encuentra en modo access point. El arreglo de memoria de las dos unidades se encuentra representado en la tabla **4-4**,

obteniendo un total de 6 nodos en la unidad WA_R1_2 y un total de 4 nodos en la unidad WA_R1_1.

Algoritmo 2 Actualización del historial

```
1:  Procedimiento: ACTUALIZACIÓN
2:      goto evaluar.
3:      leerSensores:
4:      informaciónSensores ← obtenerDistancia de sensores.
5:      if informaciónSensores == True then
6:          tempArray.x ← odometra.x
7:          tempArray.y ← odometra.y
8:          orientacin ← compasDigital.orientacin
9:          if [tempArray.x, tempArray.y] ∈ datos then
10:             actualiza(rutasPendientes)
11:         colisin ← informacinSensores.ID_colisin.
12:         if colisin.sensor_(2 o 3 o 4) == True then
13:             tempArray.rutasPosibles ← orientacin.fontral.
14:         if colisin.sensor_(0 y 1) == False then
15:             tempArray.rutasPosibles ← orientacin.laterals.
16:         datos.append(tempArray)
17:     goto leerSensores.
18:     evaluar:
19:     N ← vecinosCercanosRSSI.
20:     nodoCercanoRSSI ← −∞.
21:     nodoCercanoSSID ← null.
22:     for (i = 0; i < len(N); i++) do
23:         temRSSI = N(i).
24:         if temRSSI > nodoCercanoRSSI then
25:             nodoCercanoRSSI ← temRSSI.
26:             nodoCercanoSSID ← N(i).SSID.
27:     if nodoCercanoRSSI.lengthData > lengthData then
28:         datos ← nodoCercanoRSSI.datos.
29:     else if nodoCercanoRSSI.lengthData < lengthData then
30:         nodoCercanoRSSI.datos ← datos.
31:     goto leerSensores.
```

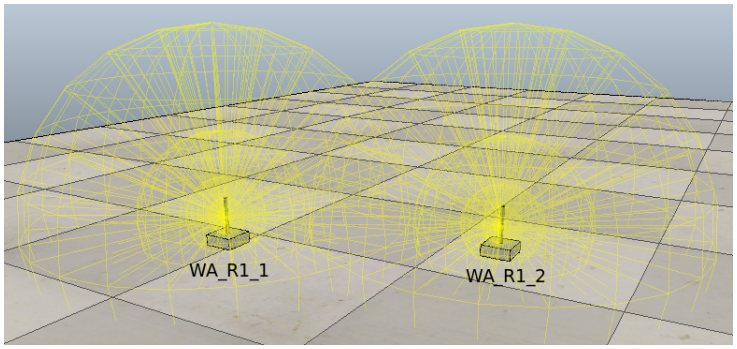


Figura 4-4.: Intercambio de información entre dos agentes.

Tabla 4-4.: Inicialización de los arreglos Fig. **4-4**.

Arreglo-WA_R1_1				**Arreglo-WA_R1_2**			
modo: Access Point				modo: Cliente			
ID_{node}	pos_x	pos_y	$UltimoID_{node}$	ID_{node}	pos_x	pos_y	$UltimoID_{node}$
0	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
2	1	3	2	2	1	3	2
3	2	3	3	3	2	3	3
–	–	–	–	4	2	5	3
–	–	–	–	5	3	5	4

Tabla 4-5.: Actualización de los arreglos Fig. **4-4** después del intercambio de información.

Arreglo-WA_R1_1				**Arreglo-WA_R1_2**			
modo: Cliente				modo: Access Point			
ID_{node}	pos_x	pos_y	$UltimoID_{node}$	ID_{node}	pos_x	pos_y	$UltimoID_{node}$
0	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
2	1	3	2	2	1	3	2
3	2	3	3	3	2	3	3
4	2	5	3	4	2	5	3
5	3	5	4	5	3	5	4

La unidad WA_R1_2 se conecta a la unidad WA_R1_1, e inicia el proceso de intercambio de información, detectando 2 nodos más almacenados en la unidad WA_R1_2 con respecto a la unidad WA_R1_1, siendo estos transmitidos y almacenados junto con el SSID WA_R1_2. La tabla **4-5** muestra el arreglo de las dos unidades luego del intercambio de la información. Este proceso es mostrado en el algoritmo 3 (ver: **anexo 3**).

Algoritmo 3 Intercambio de información

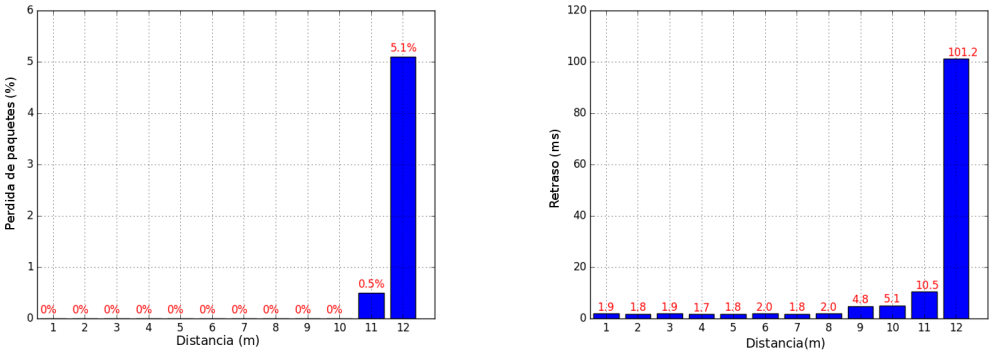
```
1:  Procedimiento: INTERCAMBIO
2:      scan:
3:      N ← vecinosCercanosRSSI.
4:      nodoCercanoRSSI ← −∞.
5:      nodoCercanoSSID ← null.
6:      for (i = 0; i < len(N); i++) do
7:          temRSSI = N(i).
8:          if temRSSI > nodoCercanoRSSI then
9:              nodoCercanoRSSI ← temRSSI.
10:             nodoCercanoSSID ← N(i).SSID.
11:     if nodoCercanoRSSI.longitudInformacin > longitudInformacin then
12:         datos ← nodoCercanoRSSI.datos.
13:     else if nodoCercanoRSSI.longitudInformacin < longitudInformacin then
14:         nodoCercanoRSSI.datos ← datos.
15:     goto scan.
```

4.3. PRUEBA DEL SISTEMA

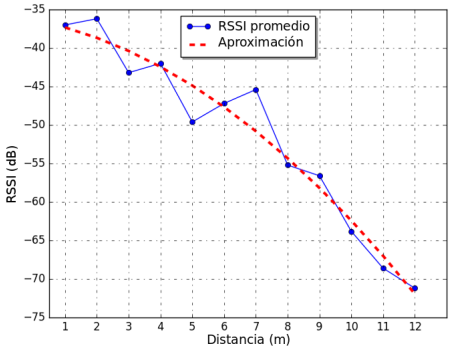
Previo al proceso de validación, es necesario modelar el uso del patrón del RSSI. Para esto, se evalúa el comportamiento del modulo inalámbrico ESP8266, el cual cumple con las características de configuración como cliente o access point. El comportamiento es evaluado enviando 100 paquetes de 1024 bytes cada uno, cambiando la distancia de transferencia de información hasta que ocurran perdidas de paquetes.

Al realizar las mediciones, se observa que las perdidas ocurren a partir de tener un RSSI menor a los $-64dB$ Fig. 4.3, además, los paquetes tienen una latencia menor a $1{,}88ms$ cuando se tiene una RSSI menor a los $-50dB$ Fig. 4.3, y al usar la medida de distancia de la evaluación y la relación del RSSI Fig. 4.3, se obtiene una distancia igual o menor a los $5m$. La validación del método propuesto para la localización y generación de mapas en entornos desconocidos on-line, es realizada a partir de tres experimentos.

El primer experimento hace uso del método propuesto mostrando la capacidad de la generación del mapa de manera descentralizada sin la necesidad de la instalación de los WA a-priori en el entorno. El segundo experimento hace uso de una unidad central para la generación del mapa, comparando el tiempo requerido contra el primer experimento. En el tercer experimento busca evaluar la capacidad de recuperación a fallos del sistema propuesto frente a un sistema centralizado. Los tres experimentos se modelan en un entorno con un área de $25m \times 25m$ Fig. **4-6** bajo el simulador V-Rep, usando un agente robótico diferencial con una velocidad constante de 15m/s. Los módulos inalámbricos de los agentes RA y WA del sistema, son modelados con un patrón de radiación omnidireccional de $5m$, sin tener en cuenta las pérdidas de la señal por obstáculos.



(a) (b)



(c)

Figura 4-5.: Evaluación del modulo wi-fi ESP8266 (**a**) paquetes perdidos; (**b**) retraso promedio; (**c**) relación distancia - RSSI.

El primer experimento logra construir el mapa topológico (Fig. 4.3) en 22 minutos con 20 segundos, durante el proceso de SLAM el RA instala un total de 9 WA para dar cobertura a todo el entorno Fig. 4.3. EL segundo experimento tiene como base un total de 16 nodos

pre-programados con la posición a-priori en el entorno (Fig. **4-8**), reduciendo el tiempo en la generación del mapa a 16 minutos con 43 segundos, al realizar la tarea de localización y mapeo con la ayuda de la posición de los nodos de la WSN [81] y de la unidad central para el procesamiento de los datos.

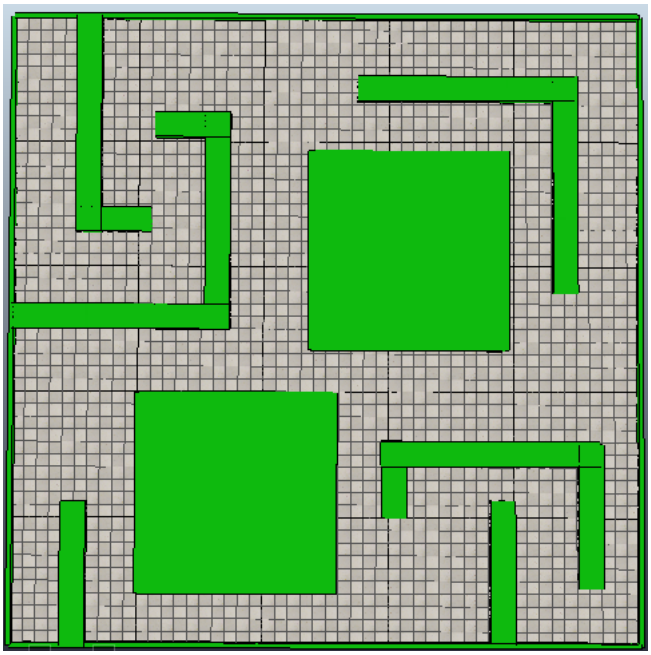
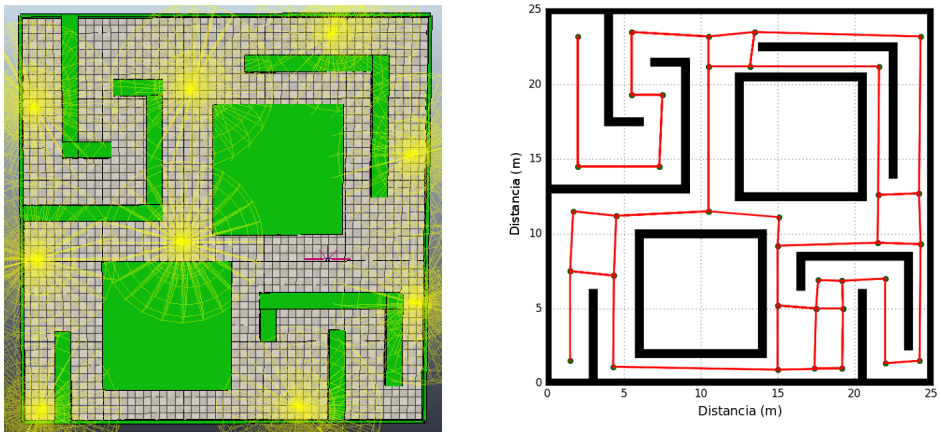


Figura 4-6.: Entorno de pruebas.



(a) (b)

Figura 4-7.: Resultados del experimento usando un sistema MAS (**a**) puesta de los nodos WA; (**b**) mapa topológico

En el tercer experimento se inyecta un error de comunicación y de funcionamiento de los motores en el RA a los 7 minutos de iniciar el experimento Fig. **4-9**. Los WA al no detectar al AR en el sistema activan un mensaje de alerta, el cual puede ser consultado por una unidad externa o por otro RA. En este caso, otro RA entra al sistema actualizando su historial acorde con los valores previos de los nodos de la red. Al final el nuevo RA logra construir el mapa topológico en un tiempo de 16 minutos con 32 segundos, para un total de 24 minutos con 21 segundos. En el caso de la sistema centralizado, se inyecta un error a la unidad central de comunicación, lo cual, evita que el sistema pueda completar la tarea de SLAM.

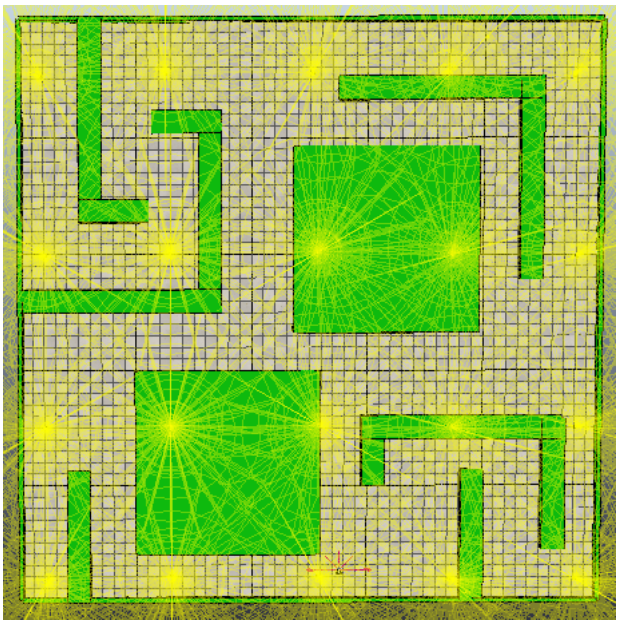
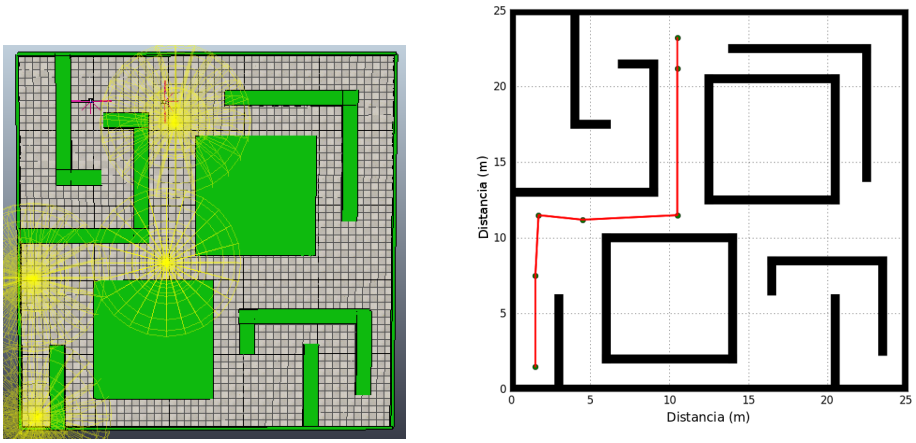


Figura 4-8.: Puesta de los sensores en un sistema centralizado.

Los anteriores experimentos muestran las características de autonomía del sistema, resaltando los siguientes puntos, los cuales son base para que un MAS sea autónomo **3-1**:

- **Modularidad:**El sistema está compuesto de múltiples nodos inalámbricos y un agente robótico autónomo, que pueden ser actualizados en cualquier momento.
- **Descentralización:**No se requiere de una unidad externa al sistema para realizar tareas de control, mapeo ni localización.
- **Procesos distribuidos:**La tarea de mapeo y localización no depende de una única unidad, el proceso de información se realiza por el RA, mientras la información de navegación esta almacenada en los nodos de la red.



(a) (b)

Figura 4-9.: Inyección de un error al RA (**a**) puesta de los nodos WA; (**b**) mapa topológico

5. CAPÍTULO 5: FRAMEWORK PARA LA DESCENTRALIZACIÓN DE SISTEMAS ROBÓTICOS MULTI-AGENTE

5.1. INTRODUCCIÓN

El modelo descentralizado en tareas de navegación presentado en el capítulo 3, demostró su aplicación para sistemas multi-agente en el capítulo 4. Sin embargo, este puede ser extendido para ser aplicado a sistemas MARS como se describe en este capítulo.

La aplicación del modelo se realiza por medio del diseño de un framework[4], en el cual se implementan agentes heterogéneos, es decir, puede ser compuesto por múltiples agentes AR y WA. Cada agente que compone el sistema puede abandonar la red o aceptar nuevas conexiones, enviando la información necesaria basándose en el historial de transferencia entre todos los nodos que compongan la red. Para lograr esto, cada agente debe cumplir con cinco procesos, de los cuales cuatro establecen la inclusión del agente al sistema, y un proceso para el envió de información los nodos más cercanos basado en el Received Signal Strength Indicator (RSSI) y el historial de datos. El framework es validado en tareas de navegación, al construir un mapa topológico de forma descentralizada en un entorno desconocido para todos los agentes que componen el sistema.

5.2. DISEÑO DEL FRAMEWORK

El diseño del framework presentado en este capítulo tiene dos aspectos importantes, primero, al plantear un sistema heterogéneo, permite la conexión de nuevas unidades sean o no agentes robóticos, y segundo, a diferencia del la aplicación en el capítulo 4, se plantea una

[4]Entorno que provee una funcionalidad generica, que puede ser aplicada para una aplicación especifica, la cual contiene modelos y metodologías de diseño

red descentralizada Ad-Hoc para la comunicación entre los agentes, evitando una conexión permanente al crear un historial de registro basado en la última transferencia de información y el indicador RSSI. Para lograr esto, es importante describir las características que deben tener los agentes que componen el MARS, junto con la descripción de los procesos del framework.

5.2.1. Descripción de los agentes

Al tener un sistema heterogéneo, se proponen dos tipos de agentes, los cuales parten del principio de los nodos de una WSN. El primer tipo son los agentes robóticos, sin importar si estos son UAV o UGV, deben tener una unidad de sensores que capture información del entorno; una unidad de control para los actuadores que permiten su desplazamiento o modificar el entorno; una unidad para procesar la información capturada o la transmitida por otro agente y, por último, una unidad inalámbrica para el envío y la recepción de información de otros agentes que compongan el sistema como se muestra en la Fig. **5-1** (ver: **anexo 5**).

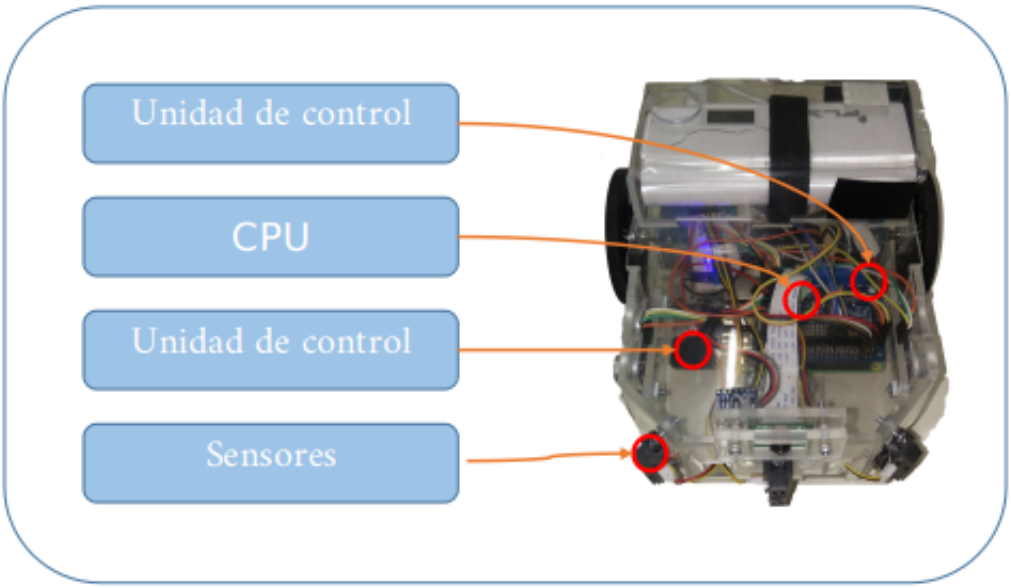


Figura 5-1.: Diagrama del agente robótico.

Estos agentes deben tener una descripción del modelo dinámico para representar su comportamiento en el sistema, esto permite el diseño de controladores para hacer tareas de formación o de localización [34], en la ecuación (5-1) se representa el modelo dinámico del robot diferencial no-holonómico usado en los experimentos de este artículo Fig. **5-2**. Este

agente al ser no-holonómico se caracteriza por no poder desplazarse directamente en los ejes (x, y), siendo el giro y el desplazamiento representados por Instantaneous Center of Curvature (ICC), donde depende del radio R y la posición (x, y) actual del agente robótico, tal que: $ICC = [x - R\sin\theta, y - R\cos\theta]$.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} x \\ y \\ \omega\delta t \end{bmatrix} \tag{5-1}$$

Donde:
$\dot{x}$ = Nueva posición eje x,
$\dot{y}$ = Nueva posición eje y,
$\dot{\theta}$ = Nueva orientación,
θ = Angulo de orientación,
ω = Angulo con respecto al punto ICC.

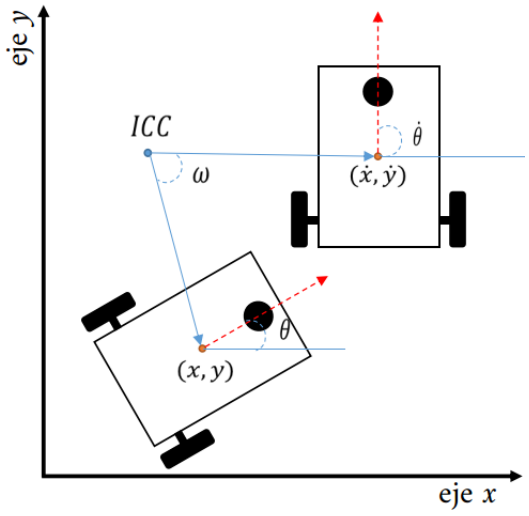


Figura 5-2.: Modelo dinámico de un robot diferencial de dos ruedas.

El segundo tipo de agente describe a las unidades de monitoreo y sensores inalámbricos, caracterizándose por tener una unidad de comunicación inalámbrica, una unidad de procesamiento y una unidad de sensores o una interfaz gráfica, dependiendo si es un nodo de WSN o un agente de monitoreo. El diagrama del agente de monitoreo usado en este artículo es mostrado en la Fig. **5-3** (ver: **anexo 6**).

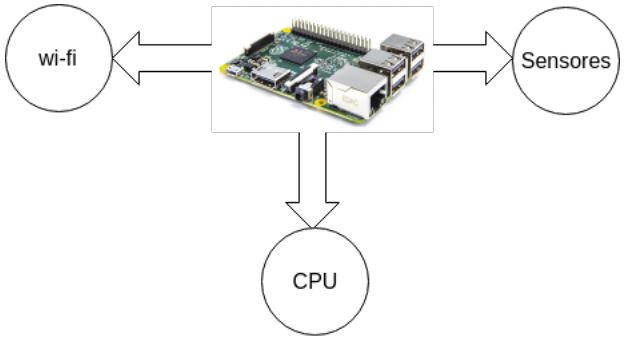


Figura 5-3.: Diagrama del agente inalámbrico o unidad de monitoreo.

Al ser implementados los agentes en el simulador V-Rep, es necesario validar los modelos cinemáticos y los sensores del agente robótico mostrado en la figura **5-1**. Se realizaron 50 pruebas en una superficie plana, arrojando los resultados por el agente robótico físico mostrados en la tabla **5-1**, mientras que los resultados arrojados por el simulador se muestran en la tabla **5-2**, mostrando un error de 0,6176 % entre los resultados del agente físico y del agente simulado.

Datos agente robótico físico			
Tiempo(s)	Distancia(m)	Velocidad(m/s)	$\omega(rad/s)$
12.3563809395	1.5	0.12	0.027

Tabla 5-1.: Arreglo dinámico del historial de información.

Datos agente robótico físico			
Tiempo(s)	Distancia(m)	Velocidad(m/s)	$\omega(rad/s)$
12.3502039909	1.49324440956	0.120908	0.0270

Tabla 5-2.: Arreglo dinámico del historial de información.

Por otro lado, los sensores de distancia se caracterizaron a partir de la toma de muestras en diferentes distancias como se muestra en la tabla **5-3**. Los anteriores datos son linealizados de manera logarítmica (ecuación 5-2) y potencial (ecuación 5-3), los cuales al ser validados en el simulador, arrojan un error del 0,05 % y del 0,06 % respectivamente, prefiriendo el modelamiento de la linealización potencial.

Datos agente robótico físico			
Valor sensor(mv)	Distancia(cm)	Valor sensor(mv)	Distancia(cm)
700.00	6	134.33	43
640.67	8	126.00	46
535.33	10	120.00	49
414.67	13	114.67	52
331.33	16	109.33	55
278.00	19	102.00	58
245.00	22	97.33	61
228.33	25	91.67	64
198.00	28	85.67	67
176.00	31	80.67	70
169.67	34	75.33	73
161.67	37	64.00	76
147.00	40	52.67	80

Tabla 5-3.: Arreglo dinámico del historial de información.

$$y = -31{,}46ln\,(x) + 201{,}79 \qquad R^2 = 0{,}9395 \tag{5-2}$$

$$y = 6579{,}5x^{-1{,}035} \qquad R^2 = 0{,}9844 \tag{5-3}$$

5.2.2. Descripción del framework

El Framework consta de cinco procesos modulares, al poder ser modificados en cualquier instante a nivel de sistema o a nivel del agente, pero al iniciar por primera vez el sistema, deben ser ejecutados de manera secuencial. El primer proceso es la caracterización de los componentes de los agentes del sistema y la relación que tienen con el entorno, el segundo es el análisis de transmisión entre las unidades, el tercero inicia a los agentes del sistema para el ingreso a la red Ad-Hoc, el cuarto proceso es la generación del historial de actividades en cada uno de los agentes del sistema, y finalmente el quinto proceso describe el envió de la información a partir de un encabezado para enviar el paquete de información. Estos cinco procesos son explicados a continuación.

Caracterización de los agentes en el sistema

Cada uno de los elementos que componen el entorno deben ser clasificados, esto es, conocer cada uno de los componentes de los agentes, el área del entorno y establecer puntos iniciales en la ubicación de nuevos agentes si es un entorno desconocido, o conocer las ubicaciones de nodos inalámbricos instalados a-priori en la escena. Para establecer los puntos iniciales de los agentes móviles en un entorno desconocido de área $D = [D_x, D_y]$, se consideran A_n agentes que componen inicialmente el MARS tal que n es un identificador numérico; cada uno con un módulo de odometría que permita procesar la distancia recorrida por medio de pulses per rotation (ppr) y el radio r de la llanta, calculando la distancia por pulso (dpp) a partir de la ecuación (5-4).

$$dpp_n = \frac{2\pi r_n}{ppr_n}. \tag{5-4}$$

Para generar una matriz de puntos de localización en el entorno, se selecciona el inferior igual a y el superior calculado por la ecuación (5-5), siendo en este caso el área del entorno Fig. **5-4**. Al generar esta grilla se permite la ubicación de los agentes en puntos conocidos facilitando las tareas de navegación [45].

$$p_{g(sup)} = \left[\frac{D_x}{max\,(dpp_n)}, \frac{D_y}{max\,(dpp_n)} \right]. \tag{5-5}$$

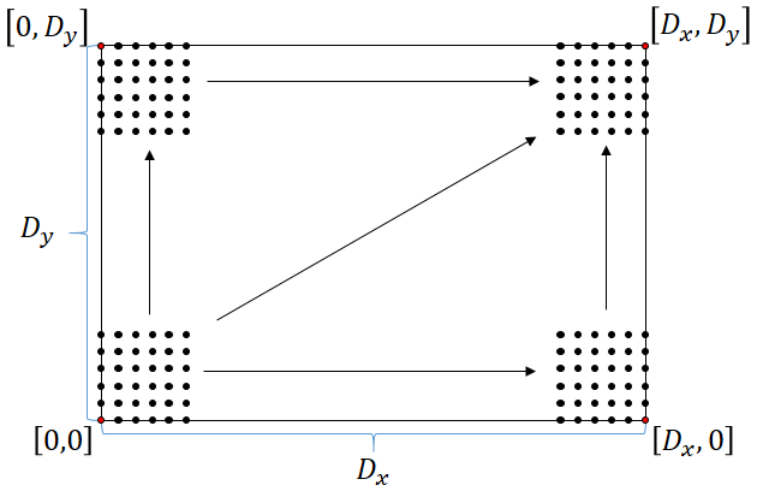


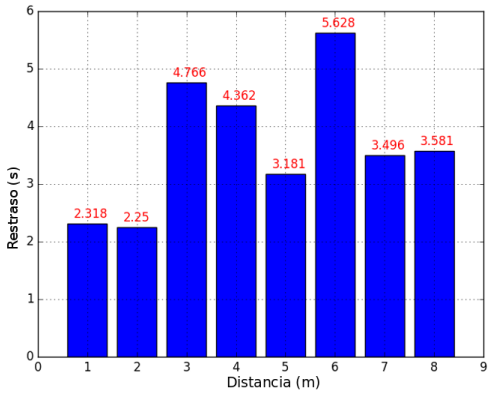
Figura 5-4.: Generación de la matriz de puntos por odometría.

Análisis de la transmisión de datos

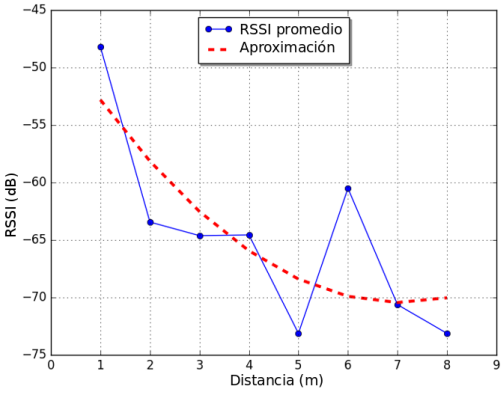
El intercambio de información de los MARS en entornos reales es un desafío, en donde, evitar la pérdida de información, aumentar tolerancia a los fallos y reducir el consumo de energía, son tareas indispensables para aumentar la autonomía de los agentes que lo componen el MARS [75, 80, 71]. El modelo de transmisión punto a punto (p2p) presenta una alta tolerancia a los fallos al no tener que mantener la comunicación con todo el sistema, evitando la dependencia de una unidad central y cuellos de botella en la comunicación del sistema, siendo ideal para ser implementado en los MARS [32]. Este modelo debe ser complementado por un protocolo de comunicación para el envió de información siendo los dos más comunes el User Datagram Protocol (UDP) y El Transmission Control Protocol (TCP).

El UDP se caracteriza por enviar paquetes de información sin necesidad de validar la correcta recepción aumentando la velocidad de transferencia pero al mismo tiempo aumentando el error del intercambio de información, en caso contrario, el TCP evita la pérdida de información al validar cada paquete enviado entre los agentes, sin embargo, esta característica presenta un inconveniente, si los agentes están con una conexión débil ocasiona una mayor pérdida de paquetes, aumentando el consumo de energía y el tiempo de transferencia al quedarse en un lazo de confirmación indefinido entre los agentes [24].

Para reducir el consumo de energía, tiempo de transmisión de paquetes y el lazo de confirmación indefinido al usar TCP [69], en el framework se propone el proceso para analizar los módulos inalámbricos usados por los agentes del MARS antes de llevarlos al entorno. En la Fig. **5-5** se muestran la caracterización del adaptador inalámbrico TL-WN823N en un entorno con un obstáculo de un muro de concreto, para esto, la fuente que se encuentra un lado del muro se ubica a una distancia inicial de un metro del receptor, esta distancia se empieza a aumentar de a un metro, y en cada cambio de ubicación de los nodos inalámbricos se inunda la red con paquetes de diferentes tamaños, esto permite analizar el RSSI, los tiempos de transmisión y la cantidad de datos perdidos y recuperados; esta separación de los nodos se realiza hasta que se presente perdida de información en la transmisión de dados, la cual en este caso fue de 8 metros (ver: **anexo 7**).



(a)



(b)

Figura 5-5.: Análisis transmisión de datos (**a**) retardo de la información; (**b**) indicador nivel RSSI

Inicialización de los agentes

Para que los agentes puedan ingresar a la red es importante que estos cumplan con el esquema de comunicación de 4 capas basado en el modelo OSI, siendo la primera capa la capa física, donde cada agente que componga el sistema heterogéneo debe tener un módulo de comunicación inalámbrica; la segunda capa es la de enlace por medio del estándar 802.11, encargada de inicializar el nombre del cliente sobre la red inalámbrica, siendo obligatorio que este sea el mismo para todos los agentes, en caso contrario, los agentes que no lo tengan no podrán compartir información en la red; la tercera capa de red se establece por el protocolo IP, siendo obligatorio la asignación de dirección única a los agentes que van a inicializar el sistema, estos deben tener la capacidad de permitir la conexión de nuevos agentes a la red, asignando una nueva dirección por medio del protocolo DHCP, para evitar colisiones

de direcciones se implementa la ecuación (5-6) para la asignación de las direcciones de los agentes iniciales y el rango de asignación de estos por medio de la ecuación (5-7), donde A es la cantidad de direcciones a asignar, N es la cantidad de agentes iniciales que componen el sistema e i es el indicador del agente tal que $i \in 0..N-1$. La cuarta y última capa es de transporte, que hace uso del protocolo TCP para garantizar la transmisión de paquetes sin pérdida de información.

$$direccion(i=0) = round\left(\frac{A+2}{N}i+2\right); direccion(i>0) = round\left(\frac{A+2}{N}i\right), \tag{5-6}$$

$$DHCP_i = \left[direccion+1\ ..\ round\left(\left(\frac{A+2}{N}(i+1)\right)+1\right)\right]. \tag{5-7}$$

Generación del historial

El historial de cada agente del sistema se representa por un arreglo dinámico (Ah_{xi}) de 4 posiciones para identificar la transferencia de la información del agente con el sistema tabla **5-4**. La primera posición del historial contiene el identificador el agente que transfiere la información, donde el rango es equivalente a la máscara de subred que se seleccione en la capa de enlace en la inicialización de los agentes; la segunda posición consiste en el identificador del agente que recibe la información que al igual que la primera posición, el rango depende de la máscara de subred; la tercera posición almacena el número de paquetes enviados entre los agentes identificados en las primeras dos posiciones, y por último, la 4 posición define los paquetes que estén pendientes para la transmisión entre los dos agentes, siendo el rango de estas dos últimas posiciones dependiente de la memoria con que dispone el agente para el almacenamiento de la información.

Arreglo de datos (Ad)			**Arreglo del historial (Ah)**				
Indice	ID	Datos	Indice	De	Para	Enviados	Pendientes

Tabla 5-4.: Arreglo dinámico del historial de información.

La actualización del historial entre dos agentes (A y B) puede ocurrir por 3 reglas como se muestra a continuación:

1. En el agente A el identificador del arreglo Ad no esté contenido en la primera casilla del historial Ah.

2. En el agente A el identificador del arreglo Ad esté contenido en la primera casilla, pero en la segunda casilla no se encuentra el identificador del agente B.

3. En el agente A el identificador del arreglo Ad esté contenido en la primera casilla y en la segunda casilla se encuentra el identificador del agente B pero en la cuarta casilla se tiene un valor mayor a cero.

Para la creación del historial en un agente se deben crear el arreglo dinámico con las 4 posiciones descritas anteriormente con una longitud inicial igual a cero, además de este arreglo, se inicializa otro arreglo dinámico unidimensional vacío para almacenar los identificadores de los cuales se ha recibido información (Ah_{xi})Para comprender como se maneja la información en el historial se propone un entorno con tres agentes robóticos identificados con Ro_1 (192.268.1.2), Ro_2 (192.268.1.85) y Ro_3 (192.268.1.170), cada uno de estos presenta un arreglo de datos almacenados (Ah_{xi}) como se muestra en la tabla **5-5**.

192.168.1.2									**192.168.1.85**									**192.168.1.170**								
Ad			**Ah**						**Ad**			**Ah**						**Ad**			**Ah**					
Indice	**ID**	**D**	**Indice**	**De**	**Para**	**E**	**Para**		**Index**	**ID**	**D**	**Index**	**De**	**Para**	**E**	**P**		**Indice**	**ID**	**D**	**Indice**	**De**	**Para**	**E**	**P**	
0	1	x0							0	2	y0							0	3	z0						
	1	x1								2	y1								3	z1						
	1	x2																	3	z2						
																			3	z3						\

Tabla 5-5.: Inicialización del historial con 3 agentes.

192.168.1.2								**192.168.1.85**							
Ad			**Ah**					**Ad**			**Ah**				
Indice	**ID**	**D**	**Indice**	**De**	**Para**	**E**	**P**	**Indice**	**ID**	**D**	**Indice**	**De**	**Para**	**E**	**P**
0	1	x0	0	1	2	3	0	0	2	y0	0	2	1	2	0
	1	x1	1	2	1	2	0		2	y1	1	1	2	3	0
	1	x2						1	1	x0					
1	2	y0							1	x1					
	2	y1							1	x2					

Tabla 5-6.: Transferencia de información entre los agentes Ro_1 y Ro_2.

Los tres agentes pertenecen a la misma red con una máscara de subred 255.255.255.0, lo cual establece que el último valor de la dirección IP sea el identificador de los agentes. La

primera transferencia de información ocurre entre los agentes Ro_1 y Ro_2, al ser la primera vez que van a transferir información entre los dos agentes, se cumple la primera regla para actualizar el historial (tabla **5-6**).

192.168.1.85								**192.168.1.170**							
Ad			**Ah**					**Ad**			**Ah**				
Indice	**ID**	**D**	**Indice**	**De**	**Para**	**E**	**P**	**Indice**	**ID**	**D**	**Indice**	**De**	**Para**	**E**	**P**
0	2	y0	0	2	1	2	0	0	3	z0	0	3	2	4	0
	2	y1	1	1	2	3	0		3	z1	1	2	3	2	0
1	1	x0	2	1	3	3	0		3	z2	2	1	3	3	0
	1	x1	3	2	3	2	0		3	z3					
	1	x2	4	3	2	4	0	1	2	y0					
2	3	z0							2	y1					
	3	z1						2	1	x0					
	3	z2							1	x1					
	3	z3							1	x2					

Tabla 5-7.: Transferencia de información entre los agentes Ro_2 y Ro_3.

La segunda transferencia ocurre entre los agentes Ro_2 y Ro_3, en este caso, los agentes Ro_2 y Ro_3 cumplen con la primera regla para la creación del historial, pero además, el agente Ro_2 cumple la segunda regla al haber realizado una transferencia previa con el agente Ro_1 (tabla **5-7**); por último, el agente Ro_2 al tener nueva información, este actualiza su historial al enviar la nueva información al agente Ro_1 (tabla **5-8**).

Encabezado del paquete de datos

El encabezado para la transmisión de paquetes de información consta de tres etapas, la primera es el código, la segunda es valor solicitado por el código (opcional), y la tercera etapa es la información solicitada por el agente. El código sirve para identificar el tipo de transmisión que se va a realizar entre los agentes robóticos, este código puede tener el valor entre el **0** y el **3**, donde el código **0** se usa para preguntar la cantidad de paquetes que tiene pendiente un agente robótico, el código **1** se usa para solicitar una cantidad determinada de paquetes, el código **2** se usa para dar inicio a la transmisión de paquetes y el código **3** es para dar envío a un paquete de forma inmediata (tabla **5-9**).

192.168.1.85								192.168.1.2							
Ad			**Ah**					**Ad**			**Ah**				
Indice	**ID**	**D**	**Indice**	**De**	**Para**	**E**	**P**	**Indice**	**ID**	**D**	**Indice**	**De**	**Para**	**E**	**P**
0	2	y0	0	2	1	2	0	0	1	x0	0	1	2	3	0
	2	y1	1	1	2	3	0		1	x1	1	2	1	2	0
1	1	x0	2	1	3	3	0		1	x2	2	3	1	4	0
	1	x1	3	2	3	2	0	1	2	y0					
	1	x2	4	3	2	4	0		2	y1					
2	3	z0	5	3	1	4	0	2	3	z0					
	3	z1							3	z1					
	3	z2							3	z2					
	3	z3							3	z3					

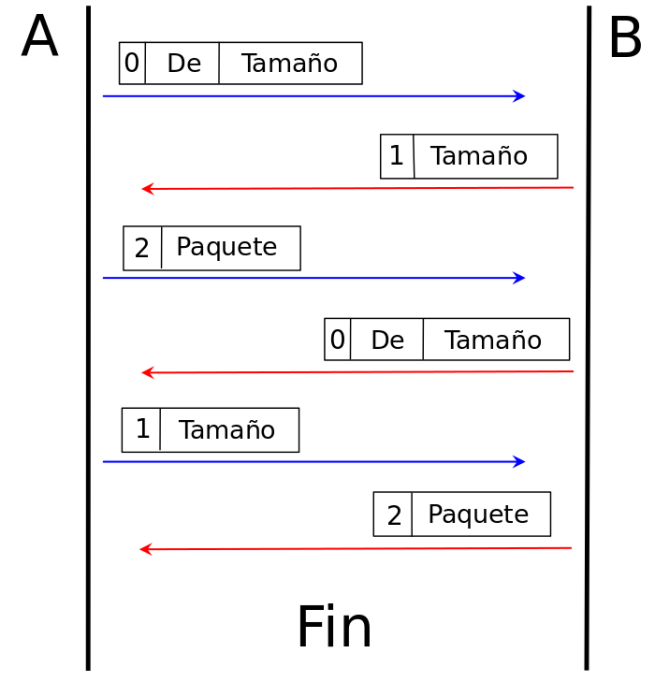
Tabla 5-8.: Transferencia de información entre los agentes Ro_2 y Ro_1, enviando la información recibida del agente Ro_3 .

Código	Descripción
0	Petición de paquetes pendientes
1	Solicitud de paquetes
2	Inicio transmisión de paquetes determinados
3	Transmisión de paquetes inmediata

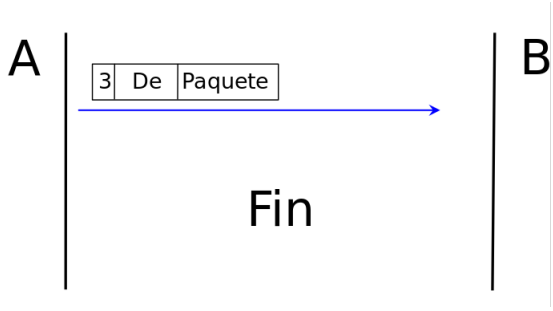
Tabla 5-9.: Códigos de identificación para la transmisión de información entre los agentes.

Acorde al código enviado al inicio del paquete, el protocolo para la transferencia de datos puede darse por solicitud de información o por envió inmediato. En la Fig. **5-6**a se puede apreciar la transferencia de datos por medio de solicitud de información entre dos agentes, el agente A envía al agente B el código **0**, proseguido el identificador de la fuente de datos, y por último la cantidad de datos que posee de la fuente con respecto al agente B; proseguido el agente B le responde al agente A con el código **1** y con la cantidad de datos que requiere de la fuente enviada con anterioridad por el agente A; por último el agente A envía el código **2** para luego enviar el paquete de datos solicitado. Lo anterior ocurre si y solo si se tiene en el historial del agente A información del agente **B**, en caso contrario, la transmisión ocurre de

manera inmediata como se muestra en la Fig. **5-6**b, siendo el código **2** utilizado y proseguido del total del paquete de datos del agente A al B.



(a)



(b)

Figura 5-6.: Análisis transmisión de datos (**a**) Transferencia de información por pedido; (**b**) Envió de información de forma inmediata

La transferencia de información entre dos agentes se da cuando se encuentren en el rango configurado de RSSI y que al tiempo alguno tenga nueva información en su historial en la columna de datos pendientes a enviar, esto se produce al agente capturar nuevos datos del entorno, o al recibir nueva información de otro agente, en este caso se usa el modo de transferencia de datos por solicitud de información; de otra forma, si el agente fuente no tiene en su historial información de envió a otro agente y este se encuentra en el rango de

cobertura, este envía de forma inmediata todo el paquete de información que contiene, este proceso se muestra en el algoritmo 4 (ver: **anexo 4**).

Algoritmo 4 Transferencia entre agentes heterogeneos.

```
1:  historial ← historialInterno
2:  for each agente ∈ agentesInCoverture do
3:      for each e ∈ historial.from.id do
4:          for each id ∈ e.to.id do
5:              banderaEnvio ← true
6:              if id = agente.id then
7:                  banderaEnvio ← false
8:                  pendiente ← historial[e][id].pendiente
9:                  if pendiente > 0 then
10:                     enviados ← historial[e][id].enviados
11:                     quanty ← sendDataRequest(enviados)
12:                     if quanty > 0 then
13:                         send(package)
14:                         historial[e][id].enviados ← enviados + pendiente
15:                 historial[e][id].enviados ← enviados + pendiente
16:                 historial[e][id].pendiente ← 0
17:             else
18:                 if banderaEnvio = true then
19:                     enviarIformacionInmediato(agente)
20:
```

5.3. APLICACIÓN DEL FRAMEWORK PARA MARS

Para crear el entorno no convexo y el modelo dinámico de los agentes robóticos se hace uso del simulador V-Rep para validar el framework descentralizado. Este simulador permite la creación de obstáculos y de agentes robóticos, los cuales se pueden controlar directamente desde Python[5] siendo creado los adaptadores inalámbricos a partir de la biblioteca Sockets. El entorno es configurado en un área de $25m^2$, generando en su interior un laberinto con celdas ubicadas a una distancia mínima de 50cm, con el fin de dar espacio suficiente para la navegación de un robot móvil.

Los agentes robóticos ubicados en este entorno deben tener la capacidad de detectar los

[5]Software disponible en `https://www.python.org/`

muros y trazar mapas topológicos de forma independiente, los cuales, deben ser compartidos con todos los agentes que compongan el sistema garantizando que cada agente tenga el mapa topológico de todo el entorno usando el framework.

Inicialmente el sistema estará compuesto por dos agentes robóticos móviles, proseguido se agregará otro agente robótico móvil, y para finalizar se incluirá un agente de monitoreo para validar la heterogeneidad del framework para aceptar diferentes tipos de agentes en tiempo real.

Al ser la primera vez que los agentes se van a ubicar en el entorno, estos deben seguir los 5 procesos del framework de forma organizada. El primer proceso es la caracterización de los agentes en el sistema, cada agente robótico simulado, se construyen bajo el modelo físico de la Fig 5-1, cada agente se ubica al extremo del entorno, de tal forma que realicen recorridos independientes. Estos agentes están equipados con encoders que permiten la lectura de odometría directamente desde el eje de los motores de manera independiente, lo cual al usar la ecuación (5-4) se tiene un generando una matriz de puntos en el entorno por la ecuación (5-5) , esto permite ubicar al robot 1 en la posición $(0{,}25m, 0{,}25m)$ o en la matriz pg(318, 318), el segundo agente robótico es ubicado en la posición $(4{,}75m, 4{,}75m)$ o en la matriz $pg(6048, 6048)$ Fig. **5-7** (ver: **anexo 8**).

El segundo proceso es el análisis de la transmisión de información, los agentes al estar equipados con un adaptador de Wi-Fi TL-WN823N, se selecciona una distancia máxima para la transmisión de datos de $2m$, es decir, cuando la red sea reconocida con un RSSI mayor a $-63{,}4375dB$, en este caso, cada dato almacenado en el arreglo dinámico estará compuesto por un byte para almacenar los vértices del grafo junto con dos flotantes de 16 bits, que representaran la posición de los vértices (x, y) del mapa topológico creado por los agentes robóticos.

El tercer proceso es la inicialización de los agentes, los cuales tienen una mascara de acceso 255.255.255.0 y una puerta de enlace con dirección IP 192.168.1.1, permitiendo la asignación de 253 agentes al sistema, para hacer la distribución de las direcciones IP en los agentes se hace uso de la ecuación (5-6), en donde para el agente $robot_1$ se le asigna la dirección IP 192.168.1.2 con un rango de asignación de direcciones DHCP desde 192.168.1.3 hasta la 192.168.1.127, y para el agente $robot_2$ es asignada la dirección IP 192.168.1.128 con un rango de asignación de direcciones DHCP desde 192.168.1.129 hasta la 192.168.1.254.

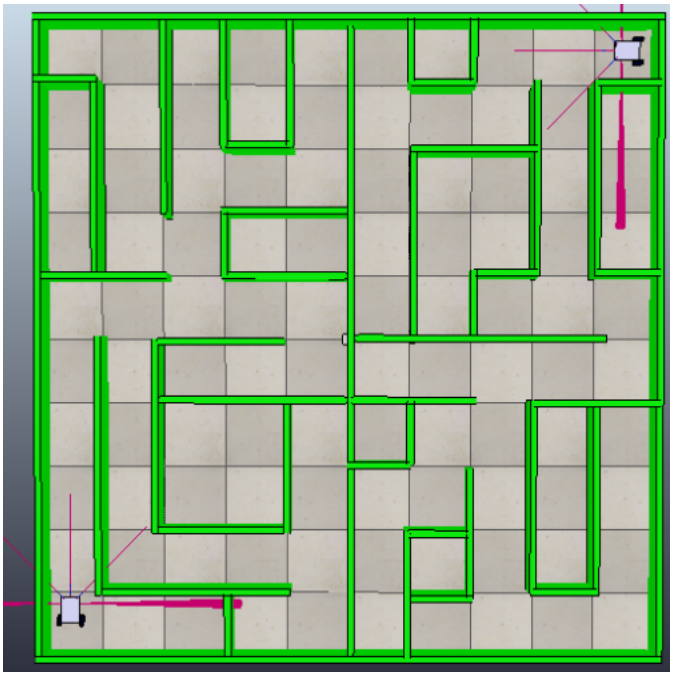


Figura 5-7.: Experimento con dos agentes robóticos.

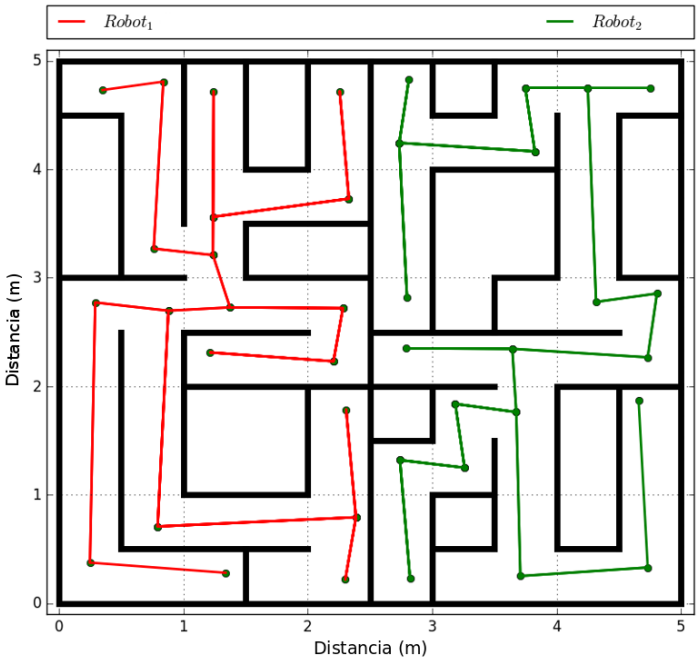


Figura 5-8.: Mapa topológico procesado por dos agentes robóticos.

El cuarto proceso es la asignación del historial a cada agente, este proceso es llevado a cabo creando un arreglo dinámico de $[x] \times [4]$ posiciones, donde x corresponde al índice dinámico que incrementa acorde a un nuevo envió o recibo de información de otro agente en el entorno. El último proceso se desarrolla en la capa de aplicación del agente.
Una vez completados los 5 procesos de los agentes en el entorno, se prosigue directamente a la implementación de la generación del mapa topológico del entorno, siendo los vértices compartidos por el modelo de comunicación del quinto proceso del framework. Este experimento se realiza comparando un sistema centralizado contra un sistema descentralizado midiendo la distancia, el tiempo consumido y la cantidad de bytes enviados para obtener todo el mapa topológico del entorno Fig. **5-8**; los resultados de los dos experimentos se pueden apreciar en la tabla 6, mostrando que un sistema centralizado al usar el esquema de comunicación del framework, presenta una reducción del 15,87 % en el tiempo de envió del mapa a los agentes contra el sistema descentralizado, sin embargo, el sistema descentralizado que hace uso completo del framework reduce la cantidad del tráfico de datos enviados para la generación del mapa en un 50 %, reduciendo el consumo de energía de los agentes robóticos [65].

Tipo	Robot IP	Distancia (m)	Tiempo (s)	Bytes enviados
Centralizado	192.168.1.2	30.5	472.46	60
	192.168.1.128	23.84	457.29	60
	192.168.1.129	–	–	120
Descentralizado	192.168.1.2	34.90	561.58	60
	192.168.1.128	28.97	535.97	60

Tabla 5-10.: Resultados del experimento con dos agentes robóticos.

Al tener caracterizados los agentes robóticos y la transmisión de los mismos, se prosigue solamente con la implementación de los procesos 3 y 4 del presente framework, teniendo en cuenta que el nuevo agente robótico se ubicara en la posición $(0{,}25m, 4{,}75m)$ o en la matriz pg(318, 6048), y los dos anteriores vuelven a ser ubicados en sus posiciones iniciales Fig **5-9**. Al agregar un nuevo agente al sistema, se prosigue con el proceso 3 para inicializar los agentes, siendo utilizada nuevamente la ecuación (4), asignado al agente $robot_1$ la dirección IP 192.168.1.2 con un rango de asignación de direcciones DHCP desde 192.168.1.3 hasta la 192.168.1.84, para el agente $robot_2$ es asignada la dirección IP 192.168.1.85 con un rango de asignación de direcciones DHCP desde 192.168.1.86 hasta la 192.168.1.169, por último al agente $robot_3$ es asignada la dirección IP 192.168.1.170 con un rango de asignación de direcciones DHCP desde 192.168.1.171 hasta la 192.168.1.254.

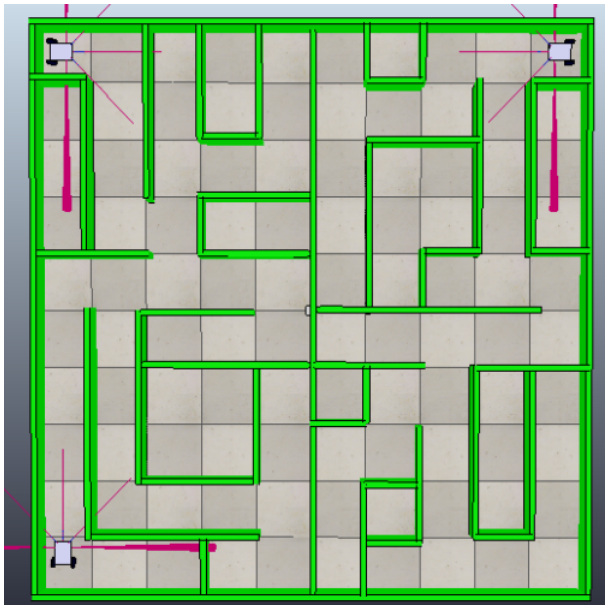


Figura 5-9.: Experimento con tres agentes robóticos.

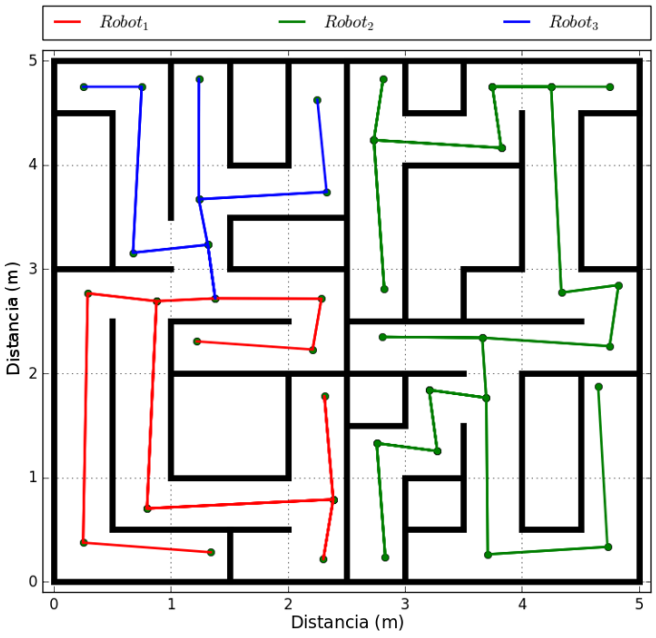


Figura 5-10.: Mapa topológico procesado por dos agentes robóticos.

El cuarto proceso solo agrega el arreglo dinámico de $[x] \times [4]$ posiciones en el agente $robot_3$, mientras que en los agentes $robot_1$ y $robot_2$ se borra el historial creado por el experimento

anterior. Nuevamente al realizar todos los procesos, se prosigue con la generación del mapa topológico Fig. **5-10**, comparando la distancia, el tiempo consumido y la cantidad de bytes enviados para obtener todo el mapa topológico del entorno, tanto en un sistema centralizado como no centralizado; los resultados de los dos experimentos se pueden apreciar en la tabla **5-11**, que igual que en el experimento con dos agentes robóticos, el sistema centralizado presenta una reducción en el tiempo del 14,55 % en el envió de la información a todo el sistema que en caso descentralizado, pero el sistema descentralizado, redujo la cantidad de bytes enviados dentro del sistema.

Tipo	Robot IP	Distancia (m)	Tiempo (s)	Bytes enviados
Centralizado	192.168.1.2	18.63	289.09	36
	192.168.1.85	23.85	447.69	60
	192.168.1.170	7.94	133.56	24
	192.168.1.86	–	–	240
Descentralizado	192.168.1.2	18.63	297.14	132
	192.168.1.85	29.20	523.93	75
	192.168.1.170	9.63	169.96	75

Tabla 5-11.: Resultados del experimento con tres agentes robóticos.

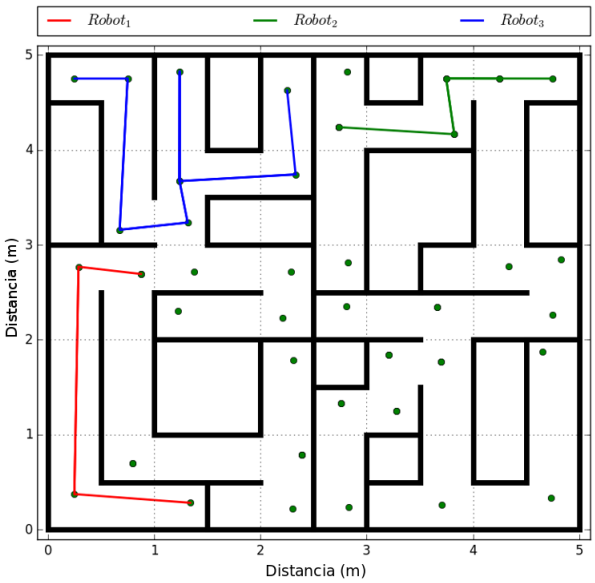


Figura 5-11.: Mapa topológico procesado por dos agentes robóticos.

Este experimento se repite haciendo que en la prueba centralizada fallara el sistema central y en la prueba descentralizada fallara el agente $robot_3$ tabla **5-12**. En la primera prueba el sistema no puede completar todo el mapa al no tener una unidad a la cual comunicar los datos y que esta los reenviara a los demás, obteniendo en cada agente un mapa parcial del entorno; caso contrario en la prueba descentralizada, en donde a pesar de la falla mecánica del agente $robot_3$ a los 110 segundos de haber iniciado la prueba, los agentes restantes son capaces de completar en su totalidad el mapa topológico del entorno; el resultado tomado del agente que presento el daño es mostrado en la Fig. **5-11**, en donde los puntos sin conexión son los vértices que faltantes.

Por otro lado, para solventar la falla mecánica del agente $robot_3$, se aprovecha la capacidad del framework de ser heterogéneo, agregando un agente de monitoreo, el cual debe cumplir las características expuestas en la sección 5.2.2. Este agente de monitoreo se implementa con el mismo modelo de adaptador inalámbrico que los agentes robóticos, por lo cual, es aplicado únicamente el cuarto proceso del framework para poder establecer la comunicación con los agentes ya instalados en el entorno. El agente de monitoreo $monitor_1$ se instala en la coordenada $(1{,}75m, 3{,}25m)$ Fig. **5-12**, esto permite que quede en el rango de envió entre los agentes $robot_3$ y $robot_1$, pero al tener mayor cobertura de conexión con el agente $robot_1$, el agente de monitoreo $monitor_1$ se conecta directamente con este y se le es asignada la dirección IP 192.168.1.3. Una vez se asigna la IP se inicia con el proceso de comunicación y de envió de datos del agente $robot_1$ al $monitor_1$, y proseguido el $monitor_1$ envía los datos al agente $robot_3$, la cantidad de bytes transmitidos por esta nueva transmisión es mostrada en la tabla **5-13**.

Tipo	Robot IP	Distancia (m)	Tiempo (s)	Bytes enviados
Centralizado	192.168.1.2	30.5	472.46	–
	192.168.1.85	23.85	447.69	–
	192.168.1.170	33.3	484.35	–
	192.168.1.86	–	–	–
Descentralizado	192.168.1.2	18.63	297.14	48
	192.168.1.85	29.20	523.93	81
	192.168.1.170	6.67	109.70	69

Tabla 5-12.: Resultados del experimento con tres agentes robóticos después de la falla del agente $robot_3$.

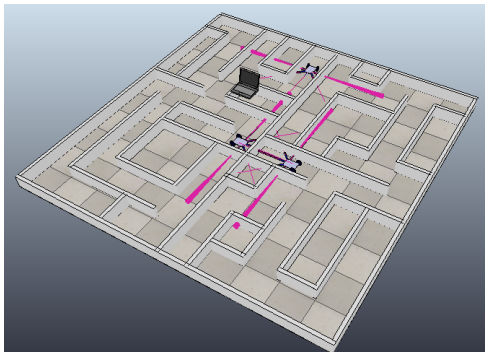


Figura 5-12.: Experimento con tres agentes robóticos y un agente de monitoreo.

Agente	Robot IP	Bytes enviados
$Robot_1$	192.168.1.85	168
$Robot_2$	192.168.1.85	81
$Robot_3$	192.168.1.170	69
$Monitor_1$	192.168.1.3	96

Tabla 5-13.: Resultado del experimento con tres agentes robóticos y una unidad de monitoreo externa.

Parte III.

REFLEXIONES

6. CAPÍTULO 6: CONCLUSIONES

A continuación se presentan las conclusiones del presente proyecto de investigación:

- El concepto y la tecnología de Redes de Sensores Inalámbricos, permitió estructurar modelos más simples para trabajar sistemas robóticos multi-agente de manera descentralizada en tareas de navegación, reduciendo el tiempo de comunicación y eliminando la dependencia del uso de una unidad central de procesamiento, logrando aumentando la autonomía del sistema en términos de modularidad y procesamiento distribuido.

- El modelo comunicación presentado no depende de una unidad principal para el procesamiento ni recolección de los datos entre los agentes que componen el sistema. Además, se elimina la necesidad de una comunicación permanente entre los agentes que componen el sistema, permitiendo la conexión en tiempo real de agentes heterogéneos sin modificar la estructura del sistema.

- Los resultados obtenidos en sistemas MAS usando el modelo propuesto, demuestran la capacidad de resolver tareas de navegación, generando un mapa topológico on-line mediante la instalación de agentes inalámbricos estáticos, apoyados en la codificación de los puntos cardinales y del procesamiento de odometría de los motores, para la localización del AR, sin la necesidad de una unidad externa o la instalación a-priori de los nodos.

- El modelo descentralizado permite que cada agente inalámbrico que compone el sistema MAS tenga una copia del mapa almacenado, por medio de la distribución de la información entre los agentes vecinos más cercanos. Durante este proceso de distribución de la información en la red, se validan el estados de los nodos, lo cual permite detectar fallas en el sistema.

- La evaluación del modelo sobre un sistema MARS en un entorno no convexo son mostrados, evidencian un mejor rendimiento del modelo propuesto frente a sistemas centralizados en la reducción del 21.67 % de datos enviados entre los agentes, la tolerancia a fallos, y en la capacidad de aceptar nuevos agentes al sistema. Sin embargo, el sistema centralizado presenta una reducción del 14.55 % en el tiempo para compartir la información con todo el sistema.

7. CAPÍTULO 7: CORRESPONDENCIA HIPÓTESIS - RESULTADOS

Con la meta de validar la hipótesis y responder la pregunta de investigación del presente trabajo de investigación, se desarrolla e implementa un nuevo modelo descentralizado para sistemas MARS, usando como base el concepto de la tecnología WSN y algoritmos de consenso. Para lograr la validación, se plantean y solucionan los diferentes objetivos específicos, logrando los siguientes resultados durante su cumplimiento:

- **Objetivo 1:** Generar un modelo de integración que establezca la relación entre los recursos y el procesamiento, a partir del concepto de WSN.
  - Se desarrollo un modelo de integración basado en el concepto de Redes de Sensores Inalámbricos en la robótica cooperativa basada en MARS, estructurando cada agente que compone el sistema como una unidad independiente más simple, y que sea capaz de interactuar con todos los nodos que componen el sistema para trabajar sistemas multi-agente de manera descentralizada, reduciendo el tiempo de comunicación y eliminando la dependencia de la unidad central, aumentando la autonomía del sistema en términos de modularidad y procesamiento distribuido.
  - Se realizo la publicación del articulo *"Decentralized model for autonomous robotic systems based on Wireless Sensor Networks"*

- **Objetivo 2:** Validar el modelo de integración en MARS en robots cooperativos.
  - Se valida el modelo propuesto, mostrando la capacidad de generar una comunicación entre todos los agentes que componen el sistema. La conexión entre los agentes se realiza unicamente cuando se obtenga nueva información del entorno, o cuando un agente actualiza la información y necesita compartirla entre todos los agentes.
  - Se realiza la validación del modelo cinemático de un agente robótico diferencial junto con la captura y envió de la información en el lenguaje Python.

- Se realizo la publicación del articulo *"Modelo Para la Simulación de Sistemas de Multi-Agentes Robóticos en Python"*

- **Objetivo 3:** Analizar el comportamiento de los agentes robóticos en ambientes dinámicos y estocásticos al usar el modelo de integración.
  - El comportamiento de los agentes robóticos es evaluado en un entorno desconocido, mostrando la capacidad de realizar tareas de navegación tales como localización y mapeo del entorno. Cada agente que compone el sistema MARS, durante la etapa de distribución de la información, puede detectar problemas o fallas en el sistema, como lo es la avería de un agente, el agente robótico puede instalar un nuevo sensor en la zona de la falla.
  - Se realizo la ponencia con título *"Autonomous multi-agent system for on-line topological mapping and location using wireless sensor network"*
  - Se realizo la publicación del articulo **Titulo:** *"Decentralized model for autonomous robotic systems based on Wireless Sensor Networks"*

- **Objetivo 4:** Diseñar un algoritmo de consenso descentralizado para agentes robóticos orientado a realizar tareas de navegación eliminando el concepto de cliente/servidor.
  - El modelo es implementado en un framework descentralizado de comunicación. En el framework se diseñan algoritmos de consenso para el intercambio de información entre los agentes del sistema sin importar si son agentes robóticos, agentes inalámbricos o agentes de monitoreo, siempre y cuando estos tengan autorización para pertenecer al sistema.
  - Se desarrollan tareas de SLAM en entornos desconocidos, eliminando el concepto cliente/servidor al hacer uso de una red Ad-Hoc, permitiendo que los agentes puedan ingresar o abandonar el sistema en cualquier momento.
  - Se realizo la publicación del articulo **Titulo:** *"A Decentralized Framework for Multi-Agent Robotic Systems"*

Lo anterior permite responder la pregunta de investigación planteada en el **capítulo 1**, al evidenciar tres elementos para aumentar la autonomía wn un sistema MARS para desarrollar tareas de navegación cooperativa en entornos dinámicos y estocásticos. El **primer** elemento es la eliminación de la unidad central para realizar el proceso y distribución de la información en la red. El **segundo** elemento es la implementación de un modelo descentralizado, que no dependa de una comunicación permanente entre todos los agentes que componen el sistema,

aumentando la tolerancia a fallos del sistema. El **tercer** y último elemento, es la aceptación de agentes heterogéneos externos al sistema inicial, aumentando la modularidad del sistema. Sin embargo, durante el desarrollo del proyecto de investigación, no se implemento el modelo en agentes robóticos físicos, esto debido a la **limitante** de los elevados costos para adquirir y diseñar un sistema MARS, evitando la validación en ambientes estocásticos no simulados.

8. CAPÍTULO 8: APORTES Y DIVULGACIONES

En este capítulo, se presentan los aportes obtenidos durante el proceso investigativo, además, se evidencian los artículos publicados y la participación en congresos internacionales.

8.1. APORTES

- Se desarrollo un modelo descentralizado para sistemas multi-agente robóticos, a partir del concepto de las redes de sensores inalámbricas. Esto permite el incremento de la autonomía de los agentes, en términos de modularidad y recuperación a fallos.
- El modelo puede ser fácilmente aplicado tanto a sistemas multi-agente como a sistemas multi-agente robóticos.
- El modelo desarrollado permite la integración de agentes heterogéneos al sistema.
- Se desarrollo un framework, que integra e implementa el modelo descentralizado, facilitando su uso en tareas de navegación usando agentes heterogéneos.
- Se elimina el termino cliente/servidor en la red creada por los agentes que componen el sistema multi-agente robótico, permitiendo el ingreso o abandono de un agente sin que afecte el funcionamiento del sistema.
- Se desarrolla un modelo para la simulación del modelo de comunicación.

8.2. REVISTAS INDEXADAS

Título	Categoría Colciencias	Enlace
A Decentralized Framework for Multi-Agent Robotic Systems	A1	http://www.mdpi.com/1424-8220/18/2/417/pdf
Decentralized Online Simultaneous Localization and Mapping for Multi-Agent Systems	A1	http://www.mdpi.com/1424-8220/18/8/2612/pdf
Methodology Design of Computational Experiments Founded on Design-based Science Case Study: Motion Models	A2	http://online-journals.org/index.php/i-joe/article/view/5850
Decentralized model for autonomous robotic systems based on Wireless Sensor Networks	B	http://www.arpnjournals.org/jeas/research_papers/rp_2016/jeas_1016_5089.pdf
Modelo Para la Simulación de Sistemas de Multi-Agentes Robóticos en Python	C	https://revistas.udistrital.edu.co/ojs/index.php/REDES/article/view/12474

Tabla 8-1.: Revistas indexadas.

8.3. EVENTOS CIENTÍFICOS

Título	Evento
Autonomous multi-agent system for on-line topological mapping and location using wireless sensor network	2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI
Analysis of TCP streaming over VANETs	Second International Conference on Research in Intelligent and Computing in Engineering (RICE-17)
Design of a System for Vehicle Traffic Estimation for Applications on IoT	MISNC '17 4th Multidisciplinary International Social Networks Conference
WSN analysis in grid topology for potato crops for IoT	MISNC '17 4th Multidisciplinary International Social Networks Conference

Tabla 8-2.: Participación en ponencias internacionales.

9. CAPÍTULO 9: TRABAJOS FUTUROS

EL modelo propuesto demostró la capacidad de realizar tareas de navegación de forma descentralizada aumentando a autonomía del sistema. Además, en este trabajo se mostró la flexibilidad del modelo al implementarlo en un framework, que permite el diseño de sistemas MARS que permite la integración con agentes que no estaban presentes al iniciar el sistema,tales como agentes inalámbricos, agentes de monitoreo u otros agentes robóticos.

La flexibilidad de integración con agentes externos al MARS permite generar nuevas investigaciones, siendo posible emplee el framework para evaluar su funcionalidad en MARS para la atención de emergencias, en especial, en tareas de búsqueda y rescate de personas en entornos de difícil acceso. Por otro lado, al ser un framework modular, se podría implementar directamente en una red de sensores, incrementando el tipo de agentes que puedan pertenecer al sistema, con el fin de hacer análisis de datos.

En el sector ambiental, el framework podría aportar en la toma de datos y control de riego, en plantaciones que no tengan una infraestructura de comunicación robusta, contemplando a los agentes robóticos como unidades estáticas para la conmutación de los actuadores de riego, permitiendo aumentar la autonomía de los nodos de la red, al reducir el número de paquetes enviados en el sistema, impactando directamente el consumo de energía de los nodos de manera positiva. Además, el uso del framework permitiría que los MARS puedan ser usados en plantas de energía de producción solar, reduciendo la actividad antrópica, haciendo más eficiente las labores de producción.

Por otro lado, el framework puede extenderse a ser aplicado en otras áreas, como el análisis en tiempo real de tráfico vehicular, teniendo en cuenta que los datos no solo deben ser distribuidos dentro del sistema, sino además, deben enviarse a una base de datos de manera remota usando el concepto de IoT como se muestra en el trabajo de Jiménez *et al.* [35].

Bibliografía

[1] Abdelhakim, Mai ; Jian Ren ; Tongtong Li: Mobile access coordinated wireless sensor networks - Topology design and throughput analysis. En: *2013 IEEE Global Communications Conference (GLOBECOM)*, 2013. – ISBN 978–1–4799–1353–4, p. 4627–4632

[2] Alismail, Hatem ; Baker, L. D. ; Browning, Brett: Continuous Trajectory estimation for 3D SLAM from actuated lidar. En: *Proceedings - IEEE International Conference on Robotics and Automation* (2014), p. 6096–6101. – ISBN 978–1–4799–3685–4

[3] Amato, Christopher ; Konidaris, George ; Cruz, Gabriel ; Maynor, Christopher a. ; How, Jonathan P. ; Kaelbling, Leslie P.: Planning for decentralized control of multiple robots under uncertainty. En: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, may 2015. – ISBN 978–1–4799–6923–4, p. 1241–1248

[4] Asama, H: Operation of cooperative multiple robots using communication in a decentralized robotic system. En: *Proc. From Perception to Action Conference* (1994), p. 36–46. ISBN 0818664827

[5] Beni, G. ; Wang, J.: Theoretical problems for the realization of distributed robotic systems. En: *Proceedings. 1991 IEEE International Conference on Robotics and Automation* (1991), Nr. April. – ISBN 0–8186–2163–X

[6] Bischoff, Rainer ; Guhl, Tim ; Wendel, Anne ; Khatami, Fariba ; Bruyninckx, Herman ; Sicilia, Bruno ; Pegman, Geoff ; Hägele, Martin ; Prassler, Erwin ; Zimmermann, Thilo ; Ibarbia, Jon A. ; Leroux, Christophe ; Tranchero, Bruno ; Labruto, Roberto ; Knoll, Alois ; Lafrenz, Reinhard ; Leuven, Katholieke U. ; Federico, Napoli ; Limited, R U R. ; Förderung, Fraunhofer-gesellschaft ; Forschung, Der ; Gesellschaft, G P S. ; Sebastián, Donostia-san: euRobotics – Shaping the future of European robotics Summary / Abstract. En: *VDE VERLAG GMBH* (2010), Nr. 4, p. 728–735. ISBN 9783800732739

[7] Brugali, Davide ; Shakhimardanov, Azamat: Component-based robotic engineering (Part II): Systems and models. En: *IEEE Robotics and Automation Magazine* 17 (2010), Nr. 1, p. 100–112. – ISBN 1070–9932

[8] Caballero, F. ; Merino, L. ; Gil, P. ; Maza, I. ; Ollero, A.: A probabilistic framework for entire WSN localization using a mobile robot. En: *Robotics and Autonomous Systems* 56 (2008), oct, Nr. 10, p. 798–806. – ISSN 09218890

[9] Cerotti, Davide ; Gribaudo, Marco ; Bobbio, Andrea: Markovian agents models for wireless sensor networks deployed in environmental protection. En: *Reliability Engineering and System Safety* 130 (2014), p. 149–158. – ISSN 09518320

[10] Chen, Wanming ; Mei, Tao ; Liang, Huawei ; You, Zhuhong ; Li, Shuai ; Meng, Max Q.-H.: Environment-Map-free Robot Navigation Based on Wireless Sensor Networks. En: *2007 International Conference on Information Acquisition*, IEEE, jul 2007. – ISBN 1–4244–1219–6, p. 569–573

[11] Cheng, Chien-fu: A new direction for solving the consensus problem in networked systems. En: *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, apr 2015. – ISBN 978–1–4673–7131–5, p. 666–670

[12] Cheng, Liying ; Xue, Dingyu ; Cong, Yang ; Jiang, Xuehui ; Zhao, Shuying: The research of environment perception based on the cooperation of multi-robot. En: *Proceedings of the 2012 24th Chinese Control and Decision Conference, CCDC 2012* (2012), p. 1914–1919. ISBN 9781457720727

[13] Cheng, Teddy M. ; Savkin, Andrey V.: Decentralized control of multi-agent systems for swarming with a given geometric pattern. En: *Computers and Mathematics with Applications* 61 (2011), Nr. 4, p. 731–744. – ISBN 0898–1221

[14] Cheng, Teddy M. ; Savkin, Andrey V. ; Javed, Faizan: Decentralized control of a group of mobile robots for deployment in sweep coverage. En: *Robotics and Autonomous Systems* 59 (2011), Nr. 7-8, p. 497–507. – ISBN 0921–8890

[15] Choi, Kiwan ; Park, Jiyoung ; Kim, Yeon-ho ; Lee, Hyoung-ki: *Chapter 3.4 - Monocular SLAM with Undelayed Initialization for an Indoor Robot1*. Vol. 60. Zhejiang University Press Co., Ltd, 2015. – 185–210 p.. – ISBN 9780128008812

[16] Corff, Sylvain L. ; Fort, Gersende: Supplement paper to "Online Expectation Maximization based algorithms for inference in hidden Markov models". 1 (2011), aug, p. 1–25

[17] Corff, Sylvain L. ; Fort, Gersende: Convergence of a Particle-based Approximation of the Block Online Expectation Maximization Algorithm. (2012), nov, p. 1–41

[18] Dogru, Sedat ; Erkmen, Aydan M. ; Erkmen, Ismet: Tracking a Sycophant Wireless Sensor Network for its seamless integration to mobile Wireless Sensor Networks. En: *2009 2nd International Conference on Adaptive Science & Technology (ICAST)* (2009), p. 110–116. ISBN 978–1–4244–3522–7

[19] Dumont, Thierry ; Le Corff, Sylvain: Simultaneous localization and mapping in wireless sensor networks. En: *Signal Processing* 101 (2014), p. 192–203. – ISSN 01651684

[20] Dunbabin, Matthew ; Marques, Lino: Robots for environmental monitoring: Significant advancements and applications. En: *IEEE Robotics and Automation Magazine* 19 (2012), Nr. February, p. 24–39. – ISBN 1070–9932 VO – 19

[21] Elena, L ; Garc, Sergio ; Barea, Rafael ; Bergasa, Luis M. ; Molinos, Eduardo J. ; Arroyo, Roberto ; Romera, Eduardo ; Pardo, Samuel: A Multi-Sensorial Simultaneous Localization and Mapping (SLAM) System for Low-Cost Micro Aerial Vehicles in GPS-Denied Environments. En: *Sensors* 17 (2017), apr, Nr. 4, p. 802. – ISSN 1424–8220

[22] Elfadil, Osman M.: Navigation algorithm for mobile robots using WSN. En: *2013 INTERNATIONAL CONFERENCE ON COMPUTING, ELECTRICAL AND ELECTRONIC ENGINEERING (ICCEEE)*, 2013. – ISBN 978–1–4673–6232–0, p. 254–259

[23] Elsts, Atis ; Selavo, Leo: A user-centric approach to wireless sensor network programming languages. En: *2012 Third International Workshop on Software Engineering for Sensor Network Applications (SESENA)* (2012), p. 29–30. ISBN 978–1–4673–1793–1

[24] Farinelli, Alessandro ; Grisetti, Giorgio ; Iocchi, Luca: Design and implementation of modular software for programming mobile robots. En: *International Journal of Advanced Robotic Systems* 3 (2006), Nr. 1, p. 037–042. – ISSN 17298806

[25] Feddema, John T. ; Lewis, Chris ; Schoenwald, David A.: Decentralized control of cooperative robotic vehicles: Theory and application. En: *IEEE Transactions on Robotics and Automation* 18 (2002), Nr. 5, p. 852–864. – ISBN 1042–296X

[26] Fierro, Rafael ; Das, Aveek ; Spletzer, John ; Esposito, Joel ; Kumar, Vijay ; Ostrowski, James P. ; Pappas, George ; Taylor, Camillo J. ; Hur, Yerang ; Alur, Rajeev ; Lee, Insup ; Grudic, Greg ; Southall, Ben: A framework and architecture for multi-robot coordination. En: *The International Journal of Robotics Research* 21 (2002), Nr. 10-11, p. 977–995. – ISBN 1301303453432

[27] Fortino, Giancarlo ; Bal, Mert ; Li, Wenfeng ; Shen, Weiming: Collaborative Wireless Sensor Networks: Architectures, Algorithms and Applications. En: *Information Fusion* 22 (2014), p. 1–2. – ISSN 15662535

[28] Furno, Lidia ; Nielsen, Mikkel C. ; Blanke, Mogens: Centralised versus decentralised control reconfiguration for collaborating underwater robots. En: *IFAC-PapersOnLine* 28 (2015), Nr. 21, p. 732–739. – ISSN 24058963

[29] Gamallo, C. ; Mucientes, M. ; Regueiro, C.V.: Omnidirectional visual SLAM under severe occlusions. En: *Robotics and Autonomous Systems* 65 (2015), p. 76–87. – ISSN 09218890

[30] García Ansola, Pablo ; De Las Morenas, Javier ; García, Andrés ; Otamendi, Javier: Distributed decision support system for airport ground handling management using WSN and MAS. En: *Engineering Applications of Artificial Intelligence* 25 (2012), p. 544–553. – ISSN 09521976

[31] IFR International Federation of Robotics. *Executive Summary World Robotics 2017 Service Robots*. 2017

[32] Iigo-Blasco, Pablo ; Diaz-Del-Rio, Fernando ; Romero-Ternero, Ma C. ; Cagigas-Muiz, Daniel ; Vicente-Diaz, Saturnino: Robotics software frameworks for multi-agent robotic systems development. En: *Robotics and Autonomous Systems* 60 (2012), Nr. 6, p. 803–821. – ISSN 09218890

[33] Innocenti, Bianca ; López, Beatriz ; Salvi, Joaquim: A multi-agent architecture with cooperative fuzzy control for a mobile robot. En: *Robotics and Autonomous Systems; Robotics and Autonomous Systems in the 50th Anniversary of Artificial Intelligence Campus Multidisciplinary in Perception and Intelligence* 55 (2007), p. 881–891. – ISSN 09218890

[34] Jha, Abhishek ; Kumar, Manoj: Two wheels differential type odometry for mobile robots. En: *Proceedings - 2014 3rd International Conference on Reliability, Infocom Technologies and Optimization: Trends and Future Directions, ICRITO 2014* (2015). ISBN 9781479968961

[35] Jiménez, Andrés ; García-Díaz, Vicente ; Anzola, John: Design of a System for Vehicle Traffic Estimation for Applications on IoT. En: *Proceedings of the 4th Multidisciplinary International Social Networks Conference on ZZZ - MISNC '17* (2017), p. 1–6. ISBN 9781450348812

[36] Kawamura, K. ; Iskarous, M.: Trends in service robots for the disabled and the elderly. En: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)* Vol. 3, IEEE, 1994. – ISBN 0–7803–1933–8, p. 1647–1654

[37] Khoshnevis, Behrokh ; Bekey, George. *Centralized sensing and control of multiple mobile robots*. 1998

[38] Koch, Olivier ; Walter, Matthew R. ; Huang, Albert S. ; Teller, Seth: Ground robot navigation using uncalibrated cameras. En: *2010 IEEE International Conference on Robotics and Automation* (2010), may, p. 2423–2430. ISBN 978–1–4244–5038–1

[39] Kwon, Ji-wook ; Kim, Jin H. ; Seo, Jiwon: Consensus-based obstacle avoidance for robotic swarm system with behavior-based control scheme. En: *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)*, IEEE, oct 2014. – ISBN 978–8–9932–1507–6, p. 751–755

[40] Lamport, Leslie: Generalized Consensus and Paxos. 2005 (April). – Informe de Investigación. – 60 p.

[41] Lee, Heon-Cheol ; Lee, Seung-Hee ; Lee, Seung-Hwan ; Lee, Tae-Seok ; Kim, Doo-Jin ; Park, Kyung-Sik ; Lee, Kong-Woo ; Lee, Beom-Hee: Comparison and analysis of scan matching techniques for Cooperative-SLAM. En: *2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)* (2011), p. 165–168. ISBN 978–1–4577–0721–6

[42] Lei, Bin ; Zeng, Liang C.: Consensus Algorithm for Swarm Robotics Cooperative Control. En: *Advanced Materials Research* 201-203 (2011), p. 1854–1857. – ISBN 1022–6680

[43] Li, Jiayuan ; Zhong, Ruofei ; Hu, Qingwu ; Ai, Mingyao: Feature-Based Laser Scan Matching and Its Application for Indoor Mapping. En: *Sensors* 16 (2016), aug, Nr. 8, p. 1265. – ISSN 1424–8220

[44] Li, Ming ; Chen, Min ; Lu, Kejie ; Mao, Shiwen ; Zhu, Hua ; Prabhakaran, B.: Robot swarm communication networks: Architectures, protocols, and applications. En: *3rd International Conference on Communications and Networking in China, ChinaCom 2008* (2008), p. 162–166. ISBN 9781424423736

[45] Liu, Tsung-Ming ; Lyons, Damian M.: Leveraging area bounds information for autonomous decentralized multi-robot exploration. En: *Robotics and Autonomous Systems* 74 (2015), dec, p. 66–78. – ISBN 0921–8890

[46] Liu, Wenhui ; Deng, Feiqi ; Liang, Jiarong ; Yan, Xuekui: Weighted average consensus problem in networks of agents with diverse time-delays. En: *Journal of Systems Engineering and Electronics* 25 (2014), Nr. 6, p. 1056–1064. – ISSN 1004–4132

[47] Mariappan, Muralindran ; Wee, Choo C. ; Vellian, Kumarheshan ; Weng, Chow K.: A navigation methodology of an holonomic mobile robot using optical tracking device (OTD). En: *TENCON 2009 - 2009 IEEE Region 10 Conference* (2009), nov, p. 1–6. ISBN 978–1–4244–4546–2

[48] Marsh, David ; Tynan, Richard ; O'Kane, Donal ; O'Hare, Gregory M P.: Autonomic wireless sensor networks. En: *Engineering Applications of Artificial Intelligence* 17 (2004), p. 741–748. – ISSN 09521976

[49] McFetridge, L. ; Ibrahim, M.Y.: A new methodology of mobile robot navigation: The agoraphilic algorithm. En: *Robotics and Computer-Integrated Manufacturing* 25 (2009), jun, Nr. 3, p. 545–551. – ISSN 07365845

[50] Mottola, Luca: Programming Wireless Sensor Networks : Fundamental Concepts and State of the Art. En: *ACM Computing Surveys* V (2011), Nr. 3, p. 1–51

[51] Ngo, Trung-dung: A Distributed Communication Protocol For Mobile Robotics Systems. En: *IFAC Proceedings Volumes (IFAC-PapersOnline)* (2014), p. 1097–1102. ISBN 9783902823625

[52] Olfati-Saber, Reza ; Murray, Richard M.: Consensus problems in networks of agents with switching topology and time-delays. En: *IEEE Transactions on Automatic Control* 49 (2004), Nr. 9, p. 1520–1533. – ISBN 0018–9286

[53] Ollero, An??bal ; Marron, Pedro J. ; Bernard, Markus ; Lepley, Jason ; La Civita, Marco ; Van Hoesel, Lodewijk ; De Andr??s, Eduardo: AWARE: Platform for autonomous self-deploying and operation of wireless sensor-actuator networks cooperating with unmanned AeRial vehiclEs. En: *SSRR2007 - IEEE International Workshop on Safety, Security and Rescue Robotics Proceedings* (2007), Nr. September. ISBN 9781424415694

[54] Ongaro, Diego ; Ousterhout, John: In Search of an Understandable Consensus Algorithm. En: *Proceedings of the 2014 USENIX Annual Technical Conference* (2014), p. 305–320. ISBN 978–1–931971–10–2

[55] Orebäck, Anders ; Christensen, Henrik I.: Evaluation of architectures for mobile robotics. En: *Autonomous Robots* 14 (2003), Nr. 1, p. 33–49. – ISSN 09295593

[56] Ota, Jun: Multi-agent robot systems as distributed autonomous systems. En: *Advanced Engineering Informatics* 20 (2006), p. 59–70. – ISSN 14740346

[57] Park, Sang H. ; Choi, Young H. ; Baek, Sang H. ; Lee, Tae K. ; Oh, Se Y.: Feature localization using neural networks for cleaning robots with ultrasonic sensors. En: *ICCAS 2007 - International Conference on Control, Automation and Systems* (2007), p. 449–453. ISBN 8995003871

[58] Paulon, a. R. ; Frohlich, a. a. ; Becker, L. B. ; Basso, F. P.: Model-driven development of WSN applications. En: *Brazilian Symposium on Computing System Engineering, SBESC* (2013), p. 161–166. – ISBN 9781479938902

[59] Paulon, A R. ; Frohlich, A. A. ; Becker, L B. ; Basso, F P.: Wireless sensor network UML profile to support model-driven development. En: *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, 2014. – ISBN 978–1–4799–4905–2, p. 227–232

[60] Qian, Huihuan ; Wu, Xinyu ; Xu, Yangsheng: *System Design*. Zhejiang University Press Co., Ltd, 2015. – 19–34 p.. – ISBN 9780128008812

[61] Ren, Wei ; Cao, Yongcan: Simulation and Experimental Study of Consensus Algorithms for Multiple Mobile Robots with Information Feedback. En: *Intelligent Automation & Soft Computing* 14 (2008), jan, Nr. 1, p. 73–87. – ISSN 1079–8587

[62] Rohrmüller, Florian ; Kourakos, Omiros ; Rambow, Matthias ; Brščić, Dražen ; Wollherr, Dirk ; Hirche, Sandra ; Buss, Martin: Interconnected performance optimization in complex robotic systems. En: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2010), oct, p. 4113–4118. ISBN 978–1–4244–6674–0

[63] Ruan, Xiaogang ; Li, Yalei ; Zhu, Xiaoqing: Kinematic parameter calibration of two-wheeled robot. En: *2012 IEEE International Conference on Mechatronics and Automation, ICMA 2012* (2012), Nr. 2, p. 81–86. ISBN 9781467312776

[64] Safa, Haidar: A novel localization algorithm for large scale wireless sensor networks. En: *Computer Communications* 45 (2014), p. 32–46. – ISSN 01403664

[65] Sahli, Nabil ; Jabeura, Nafaa ; Badra, Mohamad: Agent-based framework for sensor-to-sensor personalization. En: *Journal of Computer and System Sciences* 81 (2015), Nr. 3, p. 487–495. – ISSN 10902724

[66] Sardouk, a. ; Mansouri, M. ; Merghem-Boulahia, L. ; Gaïti, D. ; Rahim-Amoud, R.: Multi-Agent System Based Wireless Sensor Network for Crisis Management. En: *GLOBECOM 2010, 2010 IEEE Global Telecommunications Conference* (2010). – ISBN 978–1–4244–5637–6

[67] Schleicher, David ; Bergasa, Luis M. ; Oca??a, Manuel ; Barea, Rafael ; L??pez, Mar??a E.: Real-time hierarchical outdoor slam based on stereovision and GPS fusion. En: *IEEE Transactions on Intelligent Transportation Systems* 10 (2009), Nr. 3, p. 440–452. – ISBN 1524–9050

[68] Schwager, M. ; Rus, D. ; Slotine, J.-J.: Decentralized, Adaptive Coverage Control for Networked Robots. En: *The International Journal of Robotics Research* 28 (2009), Nr. 3, p. 357–375. – ISBN 1424406021

[69] Shang, Y.: On the delayed scaled consensus problems. En: *Applied Sciences (Switzerland)* 7 (2017), Nr. 7. – ISSN 20763417

[70] Shang, Yi L.: Multi-agent coordination in directed moving neighbourhood random networks. En: *Chinese Physics B* 19 (2010), Nr. 7. – ISBN 1674–1056

[71] Silva, Ricardo ; Sa Silva, Jorge ; Boavida, Fernando: Mobility in wireless sensor networks - Survey and proposal. En: *Computer Communications* 52 (2014), p. 1–20. – ISBN 01403664

[72] Stilwell, D.J.: Decentralized control synthesis for a platoon of autonomous vehicles. En: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)* Vol. 1, IEEE, 2002. – ISBN 0–7803–7272–7, p. 744–749

[73] Tang, Xian-lun ; Li, La-mei ; Jiang, Bo-jie: Mobile robot SLAM method based on multi-agent particle swarm optimized particle filter. En: *The Journal of China Universities of Posts and Telecommunications* 21 (2014), Nr. 6, p. 78–86. – ISSN 10058885

[74] Tanner, H. G. ; Christodoulakis, D. K.: Decentralized cooperative control of heterogeneous vehicle groups. En: *Robotics and Autonomous Systems* 55 (2007), Nr. 11, p. 811–823. – ISBN 0921–8890

[75] Thrun, S ; Bücken, A ; Burgard, W ; Fox, D ; Fröhlinghaus, T ; Hennig, D ; Hofmann, T ; Krell, M ; Schimdt, T: Map Learning and High-Speed Navigation in RHINO. En: Kortenkamp, David (Ed.) ; Bonasso, R P. (Ed.) ; Murphy, R (Ed.): *Artificial Intelligence and Mobile Robots*. Cambridge, MA : MIT/AAAI Press, 1997

[76] Tuna, Gurkan ; Gungor, V. C. ; Gulez, Kayhan: An autonomous wireless sensor network deployment system using mobile robots for human existence detection in case of disasters. En: *Ad Hoc Networks* 13 (2014), Nr. PART A, p. 54–68. – ISSN 15708705

[77] Tuna, Gurkan ; Gungor, Vehbi C. ; Potirakis, Stelios M. ; Zeadally, Sherali: Wireless sensor network-based communication for cooperative simultaneous localization and mapping. En: *Computers and Electrical Engineering* 41 (2015), Nr. C, p. 407–425. – ISSN 00457906

[78] Viani, Federico ; Donelli, Massimo ; Oliveri, Giacomo ; Massa, Andrea: A mobile wireless sensor network architecture for collaborative tasks achievement by means of autonomous robot swarm. En: *Antennas and Propagation Society International Symposium (APSURSI), 2010 IEEE* (2010), p. 1–4. – ISBN 978–1–4244–4967–5

[79] Wang, X. ; Chen, M. ; Kwon, T. ; Chao, H.C. *Multiple mobile agents' itinerary planning in wireless sensor networks: survey and evaluation.* 2011

[80] Wang Na ; Liu DongQian ; He KangLi: A formal description for protocols in WSN based on STeC language. En: *2013 8th International Conference on Computer Science & Education*, 2013. – ISBN 978–1–4673–4463–0, p. 921–924

[81] Xiong, Jianqiao ; Qin, Qin ; Zeng, Kemin: A Distance Measurement Wireless Localization Correction Algorithm Based on RSSI. En: *2014 Seventh International Symposium on Computational Intelligence and Design*, IEEE, dec 2014. – ISBN 978–1–4799–7004–9, p. 276–278

[82] Xu, Bin ; Kurdila, Andrew J. ; Stilwell, Daniel J.: Geometric ergodicity of the distributional consensus problem in vehicle network control. En: *Proceedings of the IEEE Conference on Decision and Control* (2010), p. 7499–7506. – ISBN 9781424477456

[83] Zhang, Fengrui ; Cao, Jianshu ; Xu, Zhenhui: An improved Particle Swarm Optimization Particle Filtering algorithm. En: *2013 International Conference on Communications, Circuits and Systems (ICCCAS)* (2013), nov, p. 173–177. ISBN 978–1–4799–3051–7

[84] Zhang, Ying ; Liu, Juan ; Hoffmann, Gabriel ; Quilling, Mark ; Payne, Kenneth ; Bose, Prasanta ; Zimdars, Andrew: Real-time indoor mapping for mobile robots with limited sensing. En: *2010 IEEE 7th International Conference on Mobile Adhoc and Sensor Systems, MASS 2010* (2010), p. 636–641. – ISBN 9781424474882

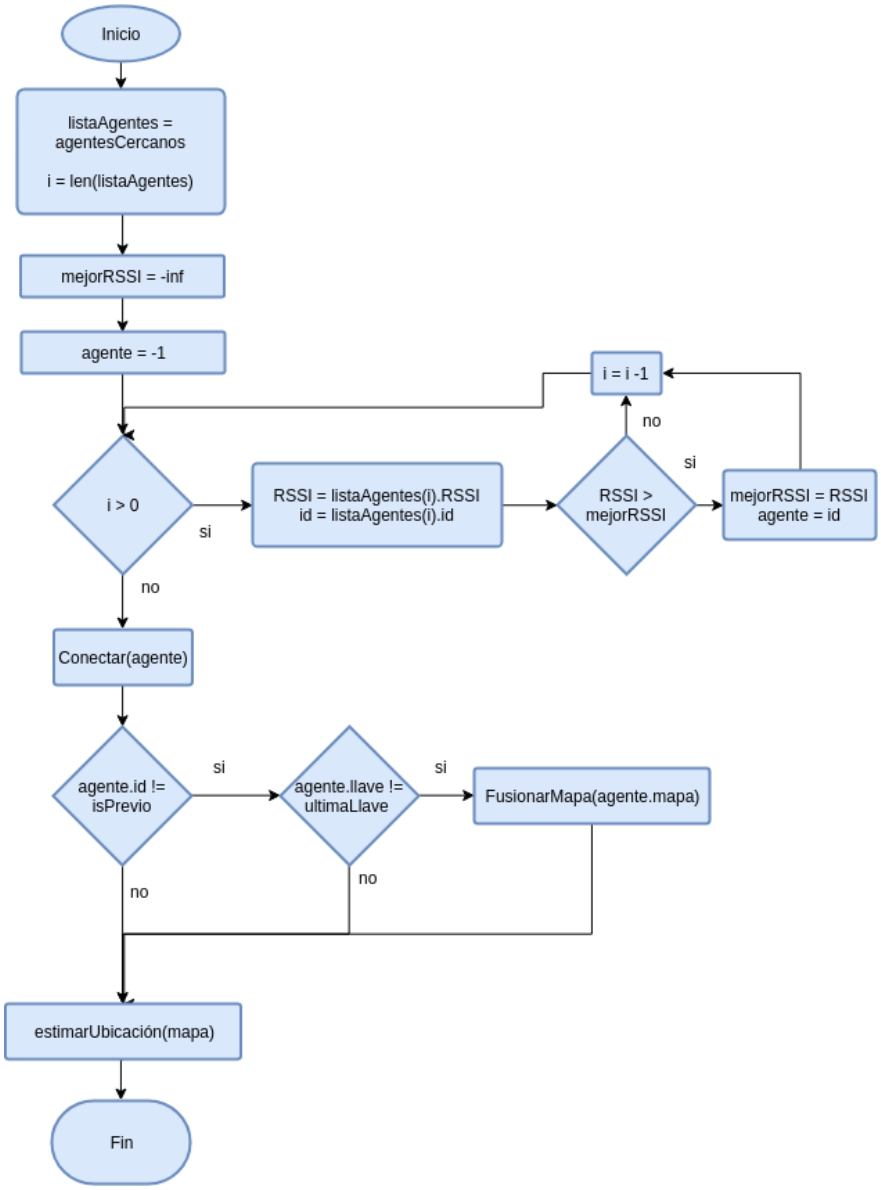
[85] Zhou, Nan ; Zhao, Xiaoguang ; Tan, Min: RSSI-based mobile robot navigation in grid-pattern wireless sensor network. En: *2013 Chinese Automation Congress* (2013), nov, p. 497–501. ISBN 978–1–4799–0333–7

[86] Zieba, Stéphane ; Polet, Philippe ; Vanderhaegen, Frédéric: Using adjustable autonomy and human-machine cooperation to make a human-machine system resilient - Application to a ground robotic system. En: *Information Sciences* 181 (2011), p. 379–397. – ISBN 0020–0255

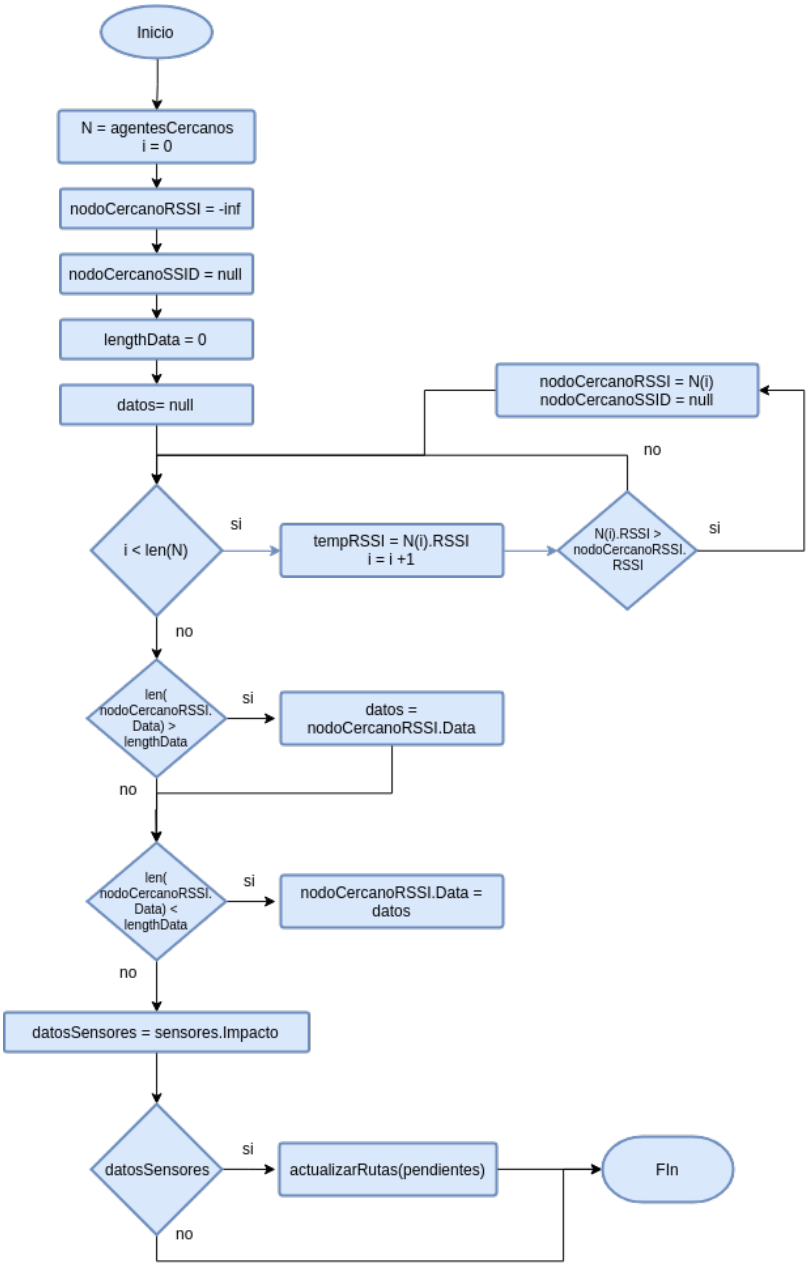
A. GLOSARIO

- **WSN:** Red de sensores que se comunican de manera inalámbrica, compuesta por una unidad para el acondicionamiento de las señales capturadas.
- **TCP:** Protocolo de comunicación usado para el envío de información, caracterizado por tener en su paquete de datos la verificación de errores para garantizar el correcto envío o recepción de la información.
- **IP:** Protocolo de información encargado de contener la dirección del host, encapsulando la información en datagramas, enviando la información de forma organizada al contener en su encabezado las dirección fuente y la de destino.
- **Agente inalámbrico:** Agente caracterizado por ser un nodo estático en una red inalámbrica, el cual interactua para capturar, almacenar, procesar y transportar la información en el sistema.
- **Agente robótico:** Agente encargado de realizar taras complejas como analizar y configurar el entorno en donde se encuentra.
- **Autonomía:**
- **Algoritmo de consenso:** Algoritmo encargado de distribuir y replicar la información entre todos los servidores que componen la red.
- **Control distribuido:** Sistema de control compuesto por múltiples agentes, que interactúan de forma coordinada para resolver problemas guiador por una unidad principal que supervisa el funcionamiento del sistema.
- **Framework:** Establece buenas practicas para crear, interpretar y analizar estructuras o arquitecturas de diseño en un dominio especifico.
- **RSSI:** Indicador que presenta la potencia de la señal de los agentes inalámbricos en una red.

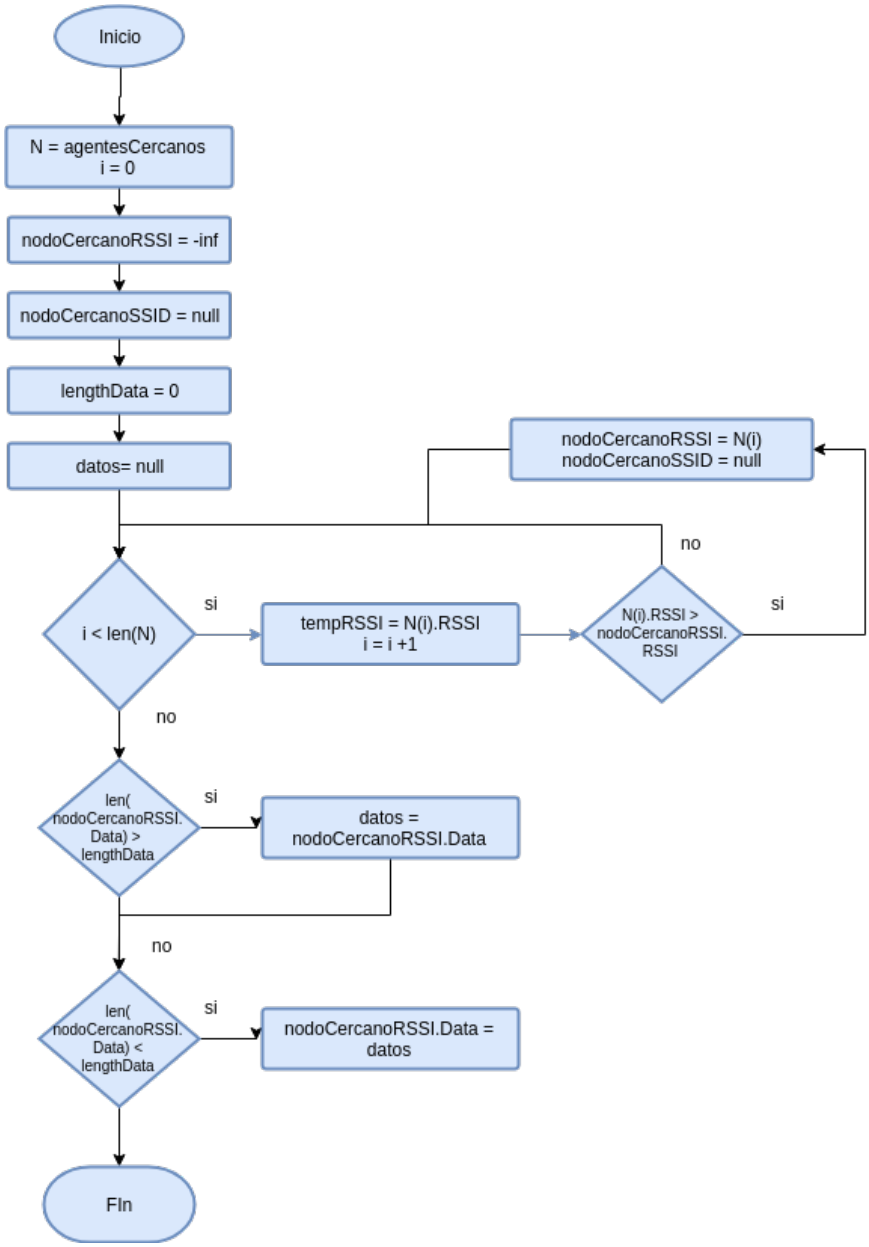
B. Anexo: Diagrama de flujo algoritmo 1



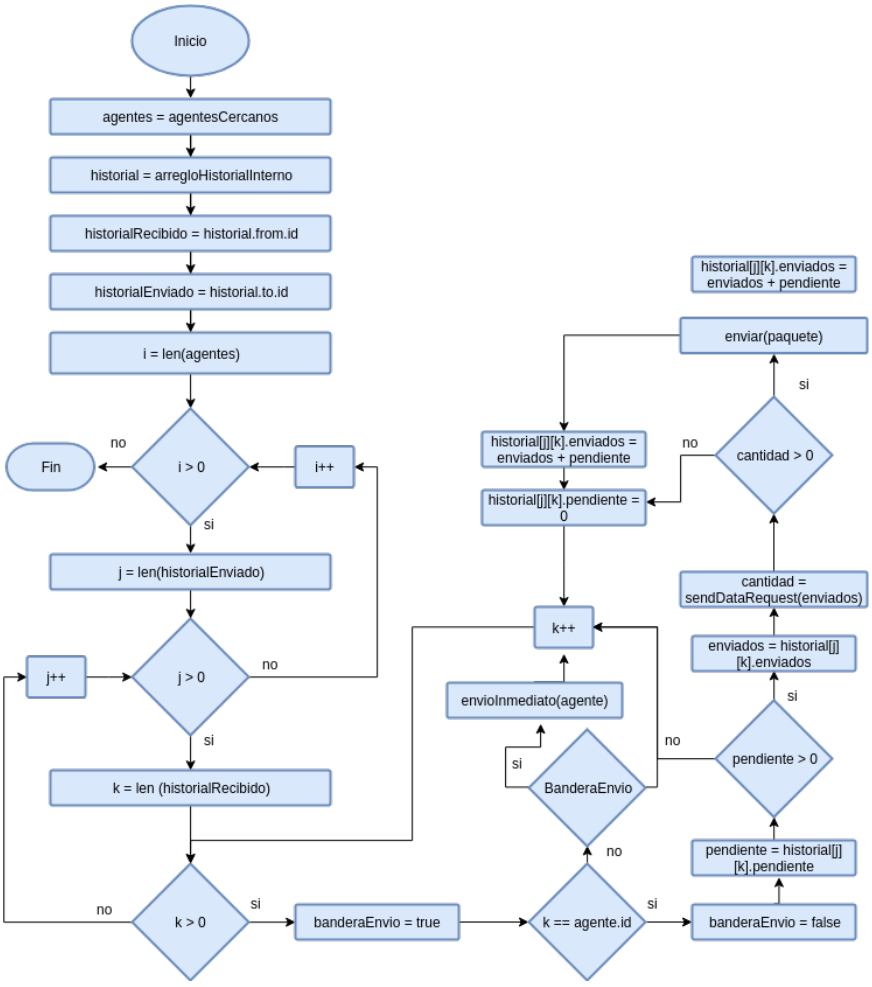
C. Anexo: Diagrama de flujo algoritmo 2



D. Anexo: Diagrama de flujo algoritmo 3



E. Anexo: Diagrama de flujo algoritmo 4



F. Anexo: Código control y monitoreo agente robótico

Listing F.1: Código principal agente robótico

```
from Distance_sensor import Distance_sensor
from  HMC5983 import HMC5983
from Adafruit_MotorHAT import Adafruit_MotorHAT, Adafruit_DCMotor
from Encoder import Encoder

import time
import atexit
import pigpio
from socket import *
import thread
import math

BUFF = 1024
HOST = '192.168.1.1'  # must be input parameter @TODO
PORT = 5000  # must be input parameter @TODO

radius = 4.5
circumference = 2 * math.pi * radius
encoder = 360.0
d_pulse = circumference / encoder


def turnOffMotors():
    mh.getMotor(1).run(Adafruit_MotorHAT.RELEASE)
    mh.getMotor(2).run(Adafruit_MotorHAT.RELEASE)


def callback(way):
    global pos_left
```

```
    pos_left += way


# print("pos motor 1={}".format(pos))

def callback2(way):
    global pos_right
    pos_right += way


# print("pos motor 2={}".format(pos2))

def move_forward(ticks):
    global sensor_data
    vel_left = 200
    vel_right = 200
    offset_right = abs(pos_right)
    offset_left = abs(pos_left)
    myMotorLeft.setSpeed(vel_left)
    myMotorRight.setSpeed(vel_right)
    myMotorLeft.run(Adafruit_MotorHAT.FORWARD)
    myMotorRight.run(Adafruit_MotorHAT.FORWARD)
    print 'moving forward'
    right_value = abs(abs(pos_right) - offset_right)
    while right_value < ticks:
        left_value = abs(abs(pos_left) - offset_left)
        right_value = abs(abs(pos_right) - offset_right)

        if right_value < left_value:
            vel_left -= 0.8#0.72
            vel_right += 0.53#>0.45
            myMotorLeft.setSpeed(int(vel_left))
            myMotorRight.setSpeed(int(vel_right))
            time.sleep(0.05)
        # myMotorRight.setSpeed(vel_right)
        elif right_value > left_value:
            vel_left += 1.2#0.65
            vel_right -= 0.4#>0.6#0.35
            myMotorLeft.setSpeed(int(vel_left))
```

```
            myMotorRight.setSpeed(int(vel_right))
            time.sleep(0.05)
            # myMotorRight.setSpeed(vel_right)
        sensor_data = sensor1.read_distance()
    pass
    print 'stop movement'
    turnOffMotors()


def move_right():
    global sensor_data
    vel_left = 110
    vel_right = 110
    offset_right = abs(pos_right)
    offset_left = abs(pos_left)
    myMotorLeft.setSpeed(vel_left)
    myMotorRight.setSpeed(vel_right)
    myMotorLeft.run(Adafruit_MotorHAT.FORWARD)
    myMotorRight.run(Adafruit_MotorHAT.BACKWARD)
    ticks = 185
    print 'moving right'
    left_value = abs(pos_left) - offset_left

    while left_value < ticks:
        left_value = abs(abs(pos_left) - offset_left)
        right_value = abs(abs(pos_right) - offset_right)
        if right_value < left_value:
            vel_left -= 0.25
            vel_right += 0.25
            myMotorLeft.setSpeed(int(vel_left))
            myMotorRight.setSpeed(int(vel_right))
            time.sleep(0.025)
        # myMotorRight.setSpeed(vel_right)
        elif right_value > left_value:
            vel_left += 0.25
            vel_right -= 0.25
            myMotorLeft.setSpeed(int(vel_left))
            myMotorRight.setSpeed(int(vel_right))
            time.sleep(0.025)
```

```
            # myMotorRight.setSpeed(vel_right)
        sensor_data = sensor1.read_distance()
    pass
    print 'stop movement'
    turnOffMotors()


def move_left():
    global sensor_data
    vel_left = 100
    vel_right = 100
    offset_right = abs(pos_right)
    offset_left = abs(pos_left)
    myMotorLeft.setSpeed(vel_left)
    myMotorRight.setSpeed(vel_right)
    myMotorLeft.run(Adafruit_MotorHAT.BACKWARD)
    myMotorRight.run(Adafruit_MotorHAT.FORWARD)
    ticks = 180
    print 'moving left'
    right_value = abs(pos_right) - offset_right
    while right_value < ticks:
        left_value = abs(abs(pos_left) - offset_left)
        right_value = abs(abs(pos_right) - offset_right)
        print left_value, right_value, offset_left, offset_right

        if right_value < left_value:
            vel_left -= 0.25
            vel_right += 0.25
            myMotorLeft.setSpeed(int(vel_left))
            myMotorRight.setSpeed(int(vel_right))
            time.sleep(0.025)
        # myMotorRight.setSpeed(vel_right)
        elif right_value > left_value:
            vel_left += 0.25
            vel_right -= 0.25
            myMotorLeft.setSpeed(int(vel_left))
            myMotorRight.setSpeed(int(vel_right))
            time.sleep(0.025)
            # myMotorRight.setSpeed(vel_right)
```

```
        sensor_data = sensor1.read_distance()
    pass
    print 'stop movement'
    turnOffMotors()


def handler(clientsock, addr):
    global flag
    global flag_movement
    global data_external
    global sensor_data
    global compass
    global myMotorLeft
    global myMotorRight
    global pos_left
    global pos_right
    while 1:
        data = clientsock.recv(BUFF)
        if not data:
            break
        # print repr(addr) + ' recv:' + repr(data)
        if len(data) > 1:
            if data[0] == 'b':
                temp_val = int(data[1:])
                ticks = temp_val / d_pulse
                data_external = ticks
                print  data_external, temp_val
                flag_movement = 1
            elif data[0] == 'f':
                temp_val = int(data[1:])
                ticks = temp_val / d_pulse
                data_external = ticks
                flag_movement = 4
            elif data[0] == 'g':
                temp_val = int(data[1:])
                ticks = temp_val / d_pulse
                data_external = ticks
                flag_movement = 5
            elif data == 'stop':
```

```
                turnOffMotors()
        else:
            # print data
            if data == 'a':
                dataTemp = ''
                for ele in sensor_data:
                    dataTemp += str(ele) + ','
                compass_data = compass.get_angle()
                dataTemp += str(compass_data[0]) + ',' +
                    str(compass_data[1]) + ',' + str(pos_left) +
                    ',' + str(pos_right)
                    clientsock.send(dataTemp)
            elif data == 'b':
                pos_left = 0
                pos_right = 0

            elif data == 'c':
                myMotorLeft.setSpeed(125)
                myMotorRight.setSpeed(125)
                myMotorLeft.run(Adafruit_MotorHAT.BACKWARD)
                myMotorRight.run(Adafruit_MotorHAT.BACKWARD)
            elif data == 'd':
                flag_movement = 3
            elif data == 'e':
                flag_movement = 2


        if "close" == data.rstrip():
            break
    flag = False
    clientsock.close()
    print addr, "- closed connection"  # log on console


# if __name__ == '__main__':
flag = True
flag_movement = 0
ADDR = (HOST, PORT)
# serversock = socket(AF_INET, SOCK_STREAM)
```

```
# serversock.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
serversock = socket()
serversock.bind(ADDR)
serversock.listen(1)

# Create the object sensor1 to capture the distance sensors
sensor1 = Distance_sensor()
# Create the object compass1 to capture the data from the compass
compass = HMC5983()
#  Create the object for control the motors
mh = Adafruit_MotorHAT(addr=0x60)
turnOffMotors()

myMotorLeft = mh.getMotor(1)
myMotorRight = mh.getMotor(2)

myMotorLeft.setSpeed(150)
myMotorRight.setSpeed(150)
pos_left = 0
pos_right = 0
pi = pigpio.pi()
decoder = Encoder(pi, 22, 23, callback2)
decoder2 = Encoder(pi, 24, 25, callback)
pos_left = 0
pos_right = 0

sensor_data = sensor1.read_distance()
# Wait for the connection
print 'waiting for connection... listening on port', PORT
clientsock, addr = serversock.accept()
print '...connected from:', addr
thread.start_new_thread(handler, (clientsock, addr))
data_external = 0
# Start the loop for reading the data of the distance sensors
while flag:

    if flag_movement == 1:
        start = time.time()
        move_forward(data_external)
```

```
        end = time.time()
        print(end - start)
        flag_movement = 0
    elif flag_movement == 2:
        move_right()
        flag_movement = 0
    elif flag_movement == 3:
        move_left()
        flag_movement = 0
    elif flag_movement == 4:
        # test to the left
        move_forward(data_external)
        time.sleep(0.5)
        move_left()
        time.sleep(0.5)
        move_forward(data_external)
        time.sleep(0.5)
        move_left()
        time.sleep(0.5)
        move_forward(data_external)
        time.sleep(0.5)
        move_left()
        time.sleep(0.5)
        move_forward(data_external)
        time.sleep(0.5)
        move_left()
        flag_movement = 0
    elif flag_movement == 5:
        # test to the right
        move_forward(data_external)
        time.sleep(0.5)
        move_right()
        time.sleep(0.5)
        move_forward(data_external)
        time.sleep(0.5)
        move_right()
        time.sleep(0.5)
        move_forward(data_external)
        time.sleep(0.5)
```

```
        move_right()
        time.sleep(0.5)
        move_forward(data_external)
        time.sleep(0.5)
        move_right()
        flag_movement = 0
    else:
        sensor_data = sensor1.read_distance()
        time.sleep(0.1)

print "Connection ended"
# Close the socket of the server
clientsock.close()
```

Listing F.2: Modulo de lectura de los sensores de distancia del agente robótico

```
#!/usr/bin/python
import smbus
import time

# ========================
# Distance_sensor_I2C Class
# Andres
#=========================
# from Distance_sensor_I2C import Distance_sensor_I2C

class Distance_sensor(object):


    def __init__(self, bus_I2C = 1):
        self.address = 0x42
        self.bus = smbus.SMBus(bus_I2C)

    def read_byte(self, adr):
        time.sleep(0.005)
        return self.bus.read_byte_data(self.address, adr)

    def read_word(self, adr):
        low = self.bus.read_byte_data(self.address, adr)
        high = self.bus.read_byte_data(self.address, adr - 1)
```

```
        val = (high << 8) + low
        return val

    def write_byte(self, adr, value):
        self.bus.write_byte_data(self.address, adr, value)

    def read_distance(self):
        #number = self.bus.read_byte(self.address)
        flag = 1
        counter_t_out = 0
        while flag == 1 and counter_t_out  <  40:
                sensor_data = []
                try:
                        sensor_data.append(self.read_byte(0x01))
                        sensor_data.append(self.read_byte(0x02))
                        sensor_data.append(self.read_byte(0x03))
                        sensor_data.append(self.read_byte(0x04))
                        sensor_data.append(self.read_byte(0x05))
                        flag = 0
                except:

                        flag = 1
                        counter_t_out  = counter_t_out +1
        return sensor_data
```

Listing F.3: Modulo de lectura de los encoder del agente robótico

```
#!/usr/bin/env python
import pigpio

class Encoder:

   def __init__(self, pi, gpioA, gpioB, callback):

      self.pi = pi
      self.gpioA = gpioA
      self.gpioB = gpioB
      self.callback = callback
      self.levA = 0
      self.levB = 0
```

```
      self.lastGpio = None
      self.pi.set_mode(gpioA, pigpio.INPUT)
      self.pi.set_mode(gpioB, pigpio.INPUT)
      self.pi.set_pull_up_down(gpioA, pigpio.PUD_UP)
      self.pi.set_pull_up_down(gpioB, pigpio.PUD_UP)
      self.cbA = self.pi.callback(gpioA, pigpio.EITHER_EDGE,
                 self._pulse)
      self.cbB = self.pi.callback(gpioB, pigpio.EITHER_EDGE,
                 self._pulse)

   def _pulse(self, gpio, level, tick):
      if gpio == self.gpioA:
         self.levA = level
      else:
         self.levB = level;
      if gpio != self.lastGpio: # debounce
         self.lastGpio = gpio

         if   gpio == self.gpioA and level == 1:
            if self.levB == 1:
               self.callback(1)
         elif gpio == self.gpioB and level == 1:
            if self.levA == 1:
               self.callback(-1)
```

Listing F.4: Modulo de contro de los motores de distancia del agente robótico

```
#!/usr/bin/python
import smbus
import time
import math
import re

# ============================
# HMC5983_I2C Class
# Andres ======================

class HMC5983(object):
    @staticmethod
    def get_bus_I2C():
```

```
        # Based from www.adafruit.com
        # http://elinux.org/RPi_HardwareHistory
        try:
            with open('/proc/cpuinfo', 'r') as infile:
                for line in infile:.
                    match = re.match('Revision\s+:\s+.*(\w{4})$'
                                                    , line)
                    if match and match.group(1) in ['0000',
                                                    '0002', '0003']:
                        return 1
                    elif match:
                        return 2
                return 0
        except:
            return 0
    __CON_REG_A = 0x00
    __CON_REG_B = 0x01
    __MODE_REGISTER = 0x02
    __AXIS_X_MSB = 0x03
    __AXIS_X_LSB = 0x04
    __AXIS_Z_MSB = 0x05
    __AXIS_Z_LSB = 0x06
    __AXIS_Y_MSB = 0x07
    __AXIS_Y_LSB = 0x08
    __STATUS_REGISTER = 0x09
    __IDENTIFICATION_REG_A = 0x0A
    __IDENTIFICATION_REG_B = 0x0B
    __IDENTIFICATION_REG_C = 0x0C
    __TEMPERATURE_MSB = 0x31
    __TEMPERATURE_LSB = 0x32

    __CONTINUOUS_MODE = 0x00
    __SINGLE_MODE = 0x01
    __IDLE_MODE = 0x02

    def __init__(self, bus_I2C = 1):
        self.address = 0x1e
        if bus_I2C != 1:
                self.bus = smbus.SMBus(HMC5983.get_bus_I2C())
```

```
        else:
                self.bus = smbus.SMBus(1)
        self.scale = 0.92
        self.write_byte(self.__CON_REG_A, 0xf0)
        # 1.3 gain LSb / Gauss 1090 (default)
        self.write_byte(self.__CON_REG_B, 0x20)
        # Set continuous mode
        self.write_byte(self.__MODE_REGISTER,
                        self.__CONTINUOUS_MODE)

    def read_byte(self, adr):
        return self.bus.read_byte_data(self.address, adr)

    def read_word(self, adr):
        low = self.bus.read_byte_data(self.address, adr)
        high = self.bus.read_byte_data(self.address, adr - 1)
        val = (high << 8) + low
        return val

    def read_word_2c(self, adr):
        val = self.read_word(adr)
        if val >= 0x8000:
            return -((65535 - val) + 1)
        else:
            return val

    def write_byte(self, adr, value):
        self.bus.write_byte_data(self.address, adr, value)

    def get_axes(self):
        flag = 1
        counter_t_out = 0
        while flag == 1 or counter_t_out  >  40:
                axes = []
                try:
                axes.append(self.read_word_2c(self.__AXIS_X_LSB)
                * self.scale) # x_axis
                axes.append(self.read_word_2c(self.__AXIS_Y_LSB)
                * self.scale)  # y_axis
```

```
                axes.append(self.read_word_2c(self.__AXIS_Z_LSB)
                * self.scale)  # z_axis
                        #print "axes =", axes
                flag = 0
                except:
                flag = 1
                counter_t_out  = counter_t_out +1
        return axes

    def get_angle(self):
        axes = self.get_axes()
        angle = math.atan2(axes[1], axes[0])
        if (angle < 0):
            angle += 2 * math.pi
        if (angle > 2*math.pi):
            angle -= 2*math.pi
        return angle,  math.degrees(angle)
```

G. Anexo: Código agente de monitoreo y de control

Listing G.1: Interfaz gráfica para el control y monitoreo del agente robótico

```
# -*- coding: utf-8 -*-

###########################################
## Python code generated with wxFormBuilder (version Dec  6 2016)
## http://www.wxformbuilder.org/
##
###########################################

import wx
import wx.xrc


###########################################
## Class FrameMain
###########################################

class FrameMain(wx.Frame):
    def __init__(self, parent):
        wx.Frame.__init__(self, parent, id=wx.ID_ANY,
                          title=u"Robot tester",
                          pos=wx.DefaultPosition,
                          size=wx.Size(622, 315),
                          style=wx.DEFAULT_FRAME_STYLE
                          | wx.TAB_TRAVERSAL)

        self.SetSizeHintsSz(wx.DefaultSize, wx.DefaultSize)

        fgSizer1 = wx.FlexGridSizer(2, 3, 0, 0)
        fgSizer1.SetFlexibleDirection(wx.BOTH)
```

```
        fgSizer1.SetNonFlexibleGrowMode(wx.FLEX_GROWMODE_SPECIFIED)

        self.m_staticText1 = wx.StaticText(self, wx.ID_ANY,
        u"Sensors", wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText1.Wrap(-1)
        fgSizer1.Add(self.m_staticText1, 0, wx.ALL |
        wx.ALIGN_CENTER_HORIZONTAL, 5)

        self.m_staticText2 = wx.StaticText(self, wx.ID_ANY,
        u"Motors", wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText2.Wrap(-1)
        fgSizer1.Add(self.m_staticText2, 0, wx.ALL
        | wx.ALIGN_CENTER_HORIZONTAL, 5)

        self.m_staticText26 = wx.StaticText(self, wx.ID_ANY,
         u"Compass", wx.DefaultPosition, wx.DefaultSize,
         wx.ALIGN_CENTRE | wx.ST_ELLIPSIZE_MIDDLE)
        self.m_staticText26.Wrap(-1)
        fgSizer1.Add(self.m_staticText26, 0, wx.ALL | wx.EXPAND, 5)

        bSizer1 = wx.BoxSizer(wx.VERTICAL)

        fgSizer2 = wx.FlexGridSizer(6, 3, 0, 0)
        fgSizer2.SetFlexibleDirection(wx.BOTH)
        fgSizer2.SetNonFlexibleGrowMode(wx.FLEX_GROWMODE_SPECIFIED)

        self.m_staticText20 = wx.StaticText(self, wx.ID_ANY,
        u"ID", wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText20.Wrap(-1)
        fgSizer2.Add(self.m_staticText20, 0, wx.ALL, 5)

        self.m_staticText21 = wx.StaticText(self, wx.ID_ANY,
        u"Distance", wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText21.Wrap(-1)
        fgSizer2.Add(self.m_staticText21, 0, wx.ALL, 5)

        self.m_staticText22 = wx.StaticText(self, wx.ID_ANY,
        u"Units", wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText22.Wrap(-1)
```

```
        fgSizer2.Add(self.m_staticText22, 0, wx.ALL, 5)

        self.m_staticText3 = wx.StaticText(self, wx.ID_ANY,
        u"0", wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText3.Wrap(-1)
        fgSizer2.Add(self.m_staticText3, 0, wx.ALL, 5)

        self.m_sensor0 = wx.StaticText(self, wx.ID_ANY,
        u"0.0", wx.DefaultPosition, wx.DefaultSize,
                                        wx.ALIGN_CENTRE |
        wx.ST_ELLIPSIZE_MIDDLE)
        self.m_sensor0.Wrap(-1)
        fgSizer2.Add(self.m_sensor0, 0, wx.ALL | wx.EXPAND, 5)

        self.m_staticText5 = wx.StaticText(self, wx.ID_ANY, u"cm",
        wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText5.Wrap(-1)
        fgSizer2.Add(self.m_staticText5, 0, wx.ALL, 5)

        self.m_staticText8 = wx.StaticText(self, wx.ID_ANY, u"1",
        wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText8.Wrap(-1)
        fgSizer2.Add(self.m_staticText8, 0, wx.ALL, 5)

        self.m_sensor1 = wx.StaticText(self, wx.ID_ANY, u"0.0",
        wx.DefaultPosition, wx.DefaultSize,
                                        wx.ALIGN_CENTRE |
        wx.ST_ELLIPSIZE_MIDDLE)
        self.m_sensor1.Wrap(-1)
        fgSizer2.Add(self.m_sensor1, 0, wx.ALL | wx.EXPAND, 5)

        self.m_staticText10 = wx.StaticText(self, wx.ID_ANY, u"cm",
        wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText10.Wrap(-1)
        fgSizer2.Add(self.m_staticText10, 0, wx.ALL, 5)

        self.m_staticText11 = wx.StaticText(self, wx.ID_ANY, u"2",
        wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText11.Wrap(-1)
```

```
        fgSizer2.Add(self.m_staticText11, 0, wx.ALL, 5)

        self.m_sensor2 = wx.StaticText(self, wx.ID_ANY, u"0.0",
        wx.DefaultPosition, wx.DefaultSize,
                                       wx.ALIGN_CENTRE |
        wx.ST_ELLIPSIZE_MIDDLE)
        self.m_sensor2.Wrap(-1)
        fgSizer2.Add(self.m_sensor2, 0, wx.ALL | wx.EXPAND, 5)

        self.m_staticText13 = wx.StaticText(self, wx.ID_ANY, u"cm",
        wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText13.Wrap(-1)
        fgSizer2.Add(self.m_staticText13, 0, wx.ALL, 5)

        self.m_staticText14 = wx.StaticText(self, wx.ID_ANY, u"3",
        wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText14.Wrap(-1)
        fgSizer2.Add(self.m_staticText14, 0, wx.ALL, 5)

        self.m_sensor3 = wx.StaticText(self, wx.ID_ANY, u"0.0",
        wx.DefaultPosition, wx.DefaultSize,
                                       wx.ALIGN_CENTRE |
        wx.ST_ELLIPSIZE_MIDDLE)
        self.m_sensor3.Wrap(-1)
        fgSizer2.Add(self.m_sensor3, 0, wx.ALL | wx.EXPAND, 5)

        self.m_staticText16 = wx.StaticText(self, wx.ID_ANY, u"cm",
        wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText16.Wrap(-1)
        fgSizer2.Add(self.m_staticText16, 0, wx.ALL, 5)

        self.m_staticText17 = wx.StaticText(self, wx.ID_ANY, u"4",
        wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText17.Wrap(-1)
        fgSizer2.Add(self.m_staticText17, 0, wx.ALL, 5)

        self.m_sensor4 = wx.StaticText(self, wx.ID_ANY, u"0.0",
        wx.DefaultPosition, wx.DefaultSize,
                                       wx.ALIGN_CENTRE |
```

```
        wx.ST_ELLIPSIZE_MIDDLE)
        self.m_sensor4.Wrap(-1)
        fgSizer2.Add(self.m_sensor4, 0, wx.ALL | wx.EXPAND, 5)

        self.m_staticText19 = wx.StaticText(self, wx.ID_ANY, u"cm",
        wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_staticText19.Wrap(-1)
        fgSizer2.Add(self.m_staticText19, 0, wx.ALL, 5)

        bSizer1.Add(fgSizer2, 1, wx.EXPAND, 5)

        self.m_ip = wx.TextCtrl(self, wx.ID_ANY, u"192.168.1.1",
        wx.DefaultPosition, wx.DefaultSize, 0)
        bSizer1.Add(self.m_ip, 0, wx.ALL | wx.EXPAND, 5)

        self.b_Connect = wx.Button(self, wx.ID_ANY, u"Connect",
        wx.DefaultPosition, wx.DefaultSize, 0)
        bSizer1.Add(self.b_Connect, 0, wx.ALL | wx.EXPAND, 5)

        self.m_status = wx.StaticText(self, wx.ID_ANY, u"Status:",
        wx.DefaultPosition, wx.DefaultSize, 0)
        self.m_status.Wrap(-1)
        bSizer1.Add(self.m_status, 0, wx.ALL | wx.EXPAND, 5)

        fgSizer1.Add(bSizer1, 1, wx.EXPAND, 5)

        bSizer2 = wx.BoxSizer(wx.VERTICAL)

        gSizer3 = wx.GridSizer(5, 3, 0, 0)

        self.m_leftE = wx.TextCtrl(self, wx.ID_ANY, u"000",
        wx.DefaultPosition, wx.DefaultSize, wx.TE_CENTRE)
        gSizer3.Add(self.m_leftE, 0, wx.ALL |
        wx.ALIGN_CENTER_HORIZONTAL, 5)

        self.m_staticText24 = wx.StaticText(self, wx.ID_ANY,
        u"Left encoder", wx.DefaultPosition, wx.DefaultSize,
                                            wx.ALIGN_CENTRE |
        wx.ST_ELLIPSIZE_MIDDLE)
```

```
        self.m_staticText24.Wrap(-1)
        gSizer3.Add(self.m_staticText24, 0, wx.ALL | wx.EXPAND, 5)

        self.b_leftE = wx.Button(self, wx.ID_ANY, u"Apply",
        wx.DefaultPosition, wx.DefaultSize, 0)
        gSizer3.Add(self.b_leftE, 0, wx.ALL, 5)

        self.m_rightE = wx.TextCtrl(self, wx.ID_ANY, u"000",
        wx.DefaultPosition, wx.DefaultSize, wx.TE_CENTRE)
        gSizer3.Add(self.m_rightE, 0, wx.ALL, 5)

        self.m_staticText25 = wx.StaticText(self, wx.ID_ANY,
        u"Right encoder", wx.DefaultPosition, wx.DefaultSize,
                                            wx.ALIGN_CENTRE |
        wx.ST_ELLIPSIZE_MIDDLE)
        self.m_staticText25.Wrap(-1)
        gSizer3.Add(self.m_staticText25, 0, wx.ALL | wx.EXPAND, 5)

        self.b_rightE = wx.Button(self, wx.ID_ANY, u"Apply",
         wx.DefaultPosition, wx.DefaultSize, 0)
        gSizer3.Add(self.b_rightE, 0, wx.ALL, 5)

        self.b_45lu = wx.Button(self, wx.ID_ANY, u"45",
        wx.DefaultPosition, wx.DefaultSize, 0)
        gSizer3.Add(self.b_45lu, 0, wx.ALL | wx.EXPAND, 5)

        self.b_upmove = wx.Button(self, wx.ID_ANY, u"^",
        wx.DefaultPosition, wx.DefaultSize, 0)
        gSizer3.Add(self.b_upmove, 0, wx.ALL | wx.EXPAND, 5)

        self.b_45ru = wx.Button(self, wx.ID_ANY, u"45",
        wx.DefaultPosition, wx.DefaultSize, 0)
        gSizer3.Add(self.b_45ru, 0, wx.ALL | wx.EXPAND, 5)

        self.b_leftmove = wx.Button(self, wx.ID_ANY, u"<",
        wx.DefaultPosition, wx.DefaultSize, 0)
        gSizer3.Add(self.b_leftmove, 0, wx.ALL | wx.EXPAND, 5)

        self.m_velocity = wx.TextCtrl(self, wx.ID_ANY, u"000",
```

```
        wx.DefaultPosition, wx.DefaultSize, wx.TE_CENTRE)
        gSizer3.Add(self.m_velocity, 0, wx.ALL |
        wx.ALIGN_CENTER_HORIZONTAL | wx.EXPAND, 5)

        self.b_rightmove = wx.Button(self, wx.ID_ANY, u">",
        wx.DefaultPosition, wx.DefaultSize, 0)
        gSizer3.Add(self.b_rightmove, 0, wx.ALL | wx.EXPAND, 5)

        self.b_45lb = wx.Button(self, wx.ID_ANY, u"45",
        wx.DefaultPosition, wx.DefaultSize, 0)
        gSizer3.Add(self.b_45lb, 0, wx.ALL | wx.EXPAND, 5)

        self.b_backmove = wx.Button(self, wx.ID_ANY, u"v",
        wx.DefaultPosition, wx.DefaultSize, 0)
        gSizer3.Add(self.b_backmove, 0, wx.ALL | wx.EXPAND, 5)

        self.b_45rb = wx.Button(self, wx.ID_ANY, u"45",
        wx.DefaultPosition, wx.DefaultSize, 0)
        gSizer3.Add(self.b_45rb, 0, wx.ALL | wx.EXPAND, 5)

        bSizer2.Add(gSizer3, 1, wx.EXPAND, 5)

        fgSizer1.Add(bSizer2, 1, wx.EXPAND, 5)

        gSizer4 = wx.GridSizer(2, 2, 0, 0)

        self.m_staticText27 = wx.StaticText(self, wx.ID_ANY,
        u"Rad", wx.DefaultPosition, wx.DefaultSize,
                                            wx.ALIGN_CENTRE |
        wx.ST_ELLIPSIZE_MIDDLE)
        self.m_staticText27.Wrap(-1)
        gSizer4.Add(self.m_staticText27, 0, wx.ALL | wx.EXPAND, 5)

        self.m_rad = wx.StaticText(self, wx.ID_ANY, u"0.00",
        wx.DefaultPosition, wx.DefaultSize,
                                   wx.ALIGN_CENTRE |
        wx.ST_ELLIPSIZE_MIDDLE)
        self.m_rad.Wrap(-1)
        gSizer4.Add(self.m_rad, 0, wx.ALL | wx.EXPAND, 5)
```

```
        self.m_staticText29 = wx.StaticText(self, wx.ID_ANY,
        u"Degrees", wx.DefaultPosition, wx.DefaultSize,
                                                wx.ALIGN_CENTRE)
        self.m_staticText29.Wrap(-1)
        gSizer4.Add(self.m_staticText29, 0, wx.EXPAND |
        wx.ALIGN_CENTER_HORIZONTAL, 5)

        self.m_deg = wx.StaticText(self, wx.ID_ANY, u"0.00",
        wx.DefaultPosition, wx.DefaultSize,
                                    wx.ALIGN_CENTRE |
        wx.ST_ELLIPSIZE_MIDDLE)
        self.m_deg.Wrap(-1)
        gSizer4.Add(self.m_deg, 0, wx.EXPAND, 5)

        fgSizer1.Add(gSizer4, 1, wx.EXPAND, 5)

        self.SetSizer(fgSizer1)
        self.Layout()
        self.m_timer1 = wx.Timer()
        self.m_timer1.SetOwner(self, wx.ID_ANY)

        self.Centre(wx.BOTH)

        # Connect Events
        self.b_Connect.Bind(wx.EVT_BUTTON, self.connectIP)
        self.b_leftE.Bind(wx.EVT_BUTTON, self.send_left)
        self.b_rightE.Bind(wx.EVT_BUTTON, self.send_right)
        self.b_45lu.Bind(wx.EVT_BUTTON, self.move_45_lu)
        self.b_upmove.Bind(wx.EVT_BUTTON, self.move_up)
        self.b_45ru.Bind(wx.EVT_BUTTON, self.move_45_ru)
        self.b_leftmove.Bind(wx.EVT_BUTTON, self.move_left)
        self.b_rightmove.Bind(wx.EVT_BUTTON, self.move_right)
        self.b_45lb.Bind(wx.EVT_BUTTON, self.move_45_lb)
        self.b_backmove.Bind(wx.EVT_BUTTON, self.move_back)
        self.b_45rb.Bind(wx.EVT_BUTTON, self.move_45_rb)
        self.Bind(wx.EVT_TIMER, self.time_complete, id=wx.ID_ANY)

    def __del__(self):
```

```
        pass

    # Virtual event handlers, overide them in your derived class
    def connectIP(self, event):
        event.Skip()

    def send_left(self, event):
        event.Skip()

    def send_right(self, event):
        event.Skip()

    def move_45_lu(self, event):
        event.Skip()

    def move_up(self, event):
        event.Skip()

    def move_45_ru(self, event):
        event.Skip()

    def move_left(self, event):
        event.Skip()

    def move_right(self, event):
        event.Skip()

    def move_45_lb(self, event):
        event.Skip()

    def move_back(self, event):
        event.Skip()

    def move_45_rb(self, event):
        event.Skip()

    def time_complete(self, event):
        event.Skip()
```

Listing G.2: Código principal para el control y monitoreo

```
import wx

# import the newly created GUI file
import WindowGui
import socket
import time
from threading import Timer,Thread,Event


class CalcFrame(WindowGui.FrameMain):
    def __init__(self, parent):
        WindowGui.FrameMain.__init__(self, parent)
        self.socket = None
        self.host = None
        self.port = 5000
        self.connection_flag = False
        self.processThread = None
        self.stopFlag = None

    def time_complete(self, line):
        # print vel
        self.socket.send('a')
        data = self.socket.recv(1024)
        values = str(data).split(',')
        # print values
        self.m_sensor0.SetLabel(values[0])
        self.m_sensor1.SetLabel(values[1])
        self.m_sensor2.SetLabel(values[2])
        self.m_sensor3.SetLabel(values[3])
        self.m_sensor4.SetLabel(values[4])
        self.m_rad.SetLabel(values[5])
        self.m_deg.SetLabel(values[6])
        self.m_leftE.SetValue(values[7])
        self.m_rightE.SetValue(values[8])
        # Create a flag to control the time of the motors
        return

    def connectIP(self, event):
```

```
        # Connect with the remote unit
        # Read the text to create the socket
        if not self.m_timer1.IsRunning():
            self.host = self.m_ip.GetValue()
            try:
                self.socket = socket.socket()
                self.socket.connect((self.host, self.port))
                self.connection_flag = True
                self.m_timer1.Start(100)
                # self.stopFlag = Event()
                # self.processThread.start()
                self.b_Connect.SetLabel("Disconnect")
                self.m_status.SetLabel("Status: Connected")
                print "Connected to the host " + self.host
                 + ' ' + str(self.port)
            except IOError as e:
                print "Cant connect to the host " +
                    self.host + ' ' + str(self.port)
                print e.args
        else:
            self.socket.send("close")
            wx.MilliSleep(200)
            self.socket.close()
            self.m_timer1.Stop()
            # self.stopFlag.set()
            self.b_Connect.SetLabel("Connect")
            self.m_status.SetLabel("Status: Disconnected")
            self.connection_flag = False
            print "Connection closed"

    def send_left(self, event):
        print "send_left"

    def send_right(self, event):
        print "send_right"

    def move_45_lu(self, event):
        print "move_45_lu"
```

```
    def move_up(self, event):
        print "move_up"
        v_send = self.m_velocity.GetValue()
        print v_send[1:-1], v_send
        self.socket.send('b'+v_send)
        print "move_up"

    def move_45_ru(self, event):
        self.socket.send('b')
        print "move_45_ru"

    def move_left(self, event):
        print "move_left"
        self.socket.send('d')
        print "move_left"

    def move_right(self, event):
        print "move_right"
        self.socket.send('e')
        # wx.MilliSleep(1750)
        # self.socket.send('stop')
        print "move_right"

    def move_45_lb(self, event):
        # test to the left movement
        print "Test left"
        v_send = self.m_velocity.GetValue()
        print v_send[1:-1], v_send
        self.socket.send('f' + v_send)
        print "End test"

    def move_back(self, event):
        print "move_back"
        self.socket.send('c')
        wx.MilliSleep(2000)
        self.socket.send('stop')
        print "move_back"

    def move_45_rb(self, event):
```

```
        # test to the right movement
        print "Test right"
        v_send = self.m_velocity.GetValue()
        print v_send[1:-1], v_send
        self.socket.send('g' + v_send)
        print "End test"


app = wx.App(False)
frame = CalcFrame(None)
frame.Show(True)
# start the applications
app.MainLoop()
```

H. Anexo: Evaluación módulos de comunicación

Listing H.1: Código evaluación modulos de comunicación

```
import RSSI_Cap
import csv

# result = RSSI_Cap.get_rssi_time(essid_name = "RPiwireless",
ip = '192.168.1.1', samples = '100', sizep = 64)
result = []
# name = 'RPiwireless'
name = 'Robot_1'
ip = '192.168.4.1'
for i in range(6, 11):
    # result.append(RSSI_Cap.get_rssi_time(
        #'VodafoneMobileWiFi-ACFB27','192.168.0.1', '1000',64*i))
    # result.append(RSSI_Cap.get_rssi_time(name, ip,'1000',64 *i))
    result.append(RSSI_Cap.get_rssi_time(name,ip,'1000',pow(2,i)))
    print 'data processed with ', pow(2, i), 'bytes'
# print result

with open('./results_esp12/module.csv', 'wb') as csvfile:
    spamwriter = csv.writer(csvfile, delimiter=';',
                            quotechar='|',
                            quoting=csv.QUOTE_MINIMAL)
    spamwriter.writerow(['Quality', 'RSSI', 'Size', 'Total time',
    'Lost', 'Min', 'Avg', 'Max', 'Mdev', 'Ipg', 'Ewma'])
    for r in result:
        spamwriter.writerow(r)

#
# with open('test.csv', 'rb') as csvfile:
#     spamreader = csv.reader(csvfile,delimiter=';',quotechar='|')
```

```
#     for row in spamreader:
#         print ', '.join(row)

print 'Test ended'
```

I. Anexo: Simulación V-Rep generación de mapa

Listing I.1: Experimento generación del mapa topológico

```
import vrep
import time
import math
import threading
import numpy as np

robot1_path = []
robot2_path = []
robot3_path = []


def movement_forward(robot, distance=0.5):
    return_code_pos, pos = vrep.simxGetObjectPosition(robot[0],
    robot[1], -1, vrep.simx_opmode_streaming)
    return_code_ori, angle = vrep.simxGetObjectOrientation(robot[0],
    robot[1], -1, vrep.simx_opmode_streaming)
    while return_code_pos and return_code_ori:
        return_code_pos, pos = vrep.simxGetObjectPosition(robot[0],
    robot[1], -1, vrep.simx_opmode_streaming)
        return_code_ori, angle = vrep.simxGetObjectOrientation(
        robot[0], robot[1], -1, vrep.simx_opmode_streaming)
    angle_degrees = math.degrees(angle[2])
    if 88 <= angle_degrees <= 92:
        plane = 1
    else:
        if -92 <= angle_degrees <= -88:
            plane = 1
            distance *= -1
        else:
```

```
            if -2 <= angle_degrees <= 2:
                plane = 0
            else:
                plane = 0
                distance *= -1
    position_objective = pos[plane] + distance
    vrep.simxSetJointTargetVelocity(robot[0], robot[2],
    velocity,
    vrep.simx_opmode_streaming)
    vrep.simxSetJointTargetVelocity(robot[0], robot[3],
    velocity,
    vrep.simx_opmode_streaming)
    if distance > 0:
        while pos[plane] < position_objective:
            _, pos = vrep.simxGetObjectPosition(robot[0],
    robot[1], -1, vrep.simx_opmode_buffer)
    else:
        while pos[plane] > position_objective:
            _, pos = vrep.simxGetObjectPosition(robot[0],
    robot[1], -1, vrep.simx_opmode_buffer)

    vrep.simxSetJointTargetVelocity(robot[0], robot[2],
    0, vrep.simx_opmode_streaming)
    vrep.simxSetJointTargetVelocity(robot[0], robot[3],
    0, vrep.simx_opmode_streaming)
    return pos[0], pos[1]


def movement_right(robot):
    # movement_forward(robot, 0.0075)
    return_code_pos, pos = vrep.simxGetObjectPosition(
    robot[0], robot[1], -1, vrep.simx_opmode_streaming)
    return_code_ori, angle = vrep.simxGetObjectOrientation(
    robot[0], robot[1], -1, vrep.simx_opmode_streaming)
    while return_code_pos and return_code_ori:
        return_code_pos, pos = vrep.simxGetObjectPosition(
        robot[0], robot[1], -1, vrep.simx_opmode_streaming)
        return_code_ori, angle = vrep.simxGetObjectOrientation(
        robot[0], robot[1], -1, vrep.simx_opmode_streaming)
```

```
    angle_degrees = math.degrees(angle[2])
    flag = False
    if 88 <= angle_degrees <= 92:
        angle_objective = 0
    else:
        if -92 <= angle_degrees <= -88:
            angle_objective = 180
            flag = True
        else:
            if -2 <= angle_degrees <= 2:
                angle_objective = -90
                flag = True
            else:
                angle_objective = 90

    vrep.simxSetJointTargetVelocity(robot[0], robot[2],
    velocity * .03, vrep.simx_opmode_streaming)
    vrep.simxSetJointTargetVelocity(robot[0], robot[3],
    -velocity * .03, vrep.simx_opmode_streaming)
    if not flag:
        while (angle_objective + 2) < abs(angle_degrees):
            _, angle = vrep.simxGetObjectOrientation(robot[0],
    robot[1], -1, vrep.simx_opmode_streaming)
            angle_degrees = math.degrees(angle[2])
    else:
        while abs(angle_objective) - 2 > abs(angle_degrees):
            _, angle = vrep.simxGetObjectOrientation(robot[0],
    robot[1], -1, vrep.simx_opmode_streaming)
            angle_degrees = math.degrees(angle[2])

    vrep.simxSetJointTargetVelocity(robot[0], robot[2], 0,
     vrep.simx_opmode_streaming)
    vrep.simxSetJointTargetVelocity(robot[0], robot[3], 0,
    vrep.simx_opmode_streaming)


def movement_left(robot):
    return_code_pos, pos = vrep.simxGetObjectPosition(robot[0],
    robot[1], -1, vrep.simx_opmode_streaming)
```

```
return_code_ori, angle = vrep.simxGetObjectOrientation(
robot[0], robot[1], -1, vrep.simx_opmode_streaming)
while return_code_pos and return_code_ori:
    return_code_pos, pos = vrep.simxGetObjectPosition(
    robot[0],robot[1], -1, vrep.simx_opmode_streaming)
    return_code_ori, angle = vrep.simxGetObjectOrientation(
    robot[0],robot[1], -1, vrep.simx_opmode_streaming)
angle_degrees = math.degrees(angle[2])
flag = False
if 88 <= angle_degrees <= 92:
    angle_objective = 180
else:
    if -92 <= angle_degrees <= -88:
        angle_objective = 0
    else:
        if -2 <= angle_degrees <= 2:
            angle_objective = 90
        else:
            angle_objective = -90
            flag = True
vrep.simxSetJointTargetVelocity(robot[0], robot[2],
 -velocity * .03, vrep.simx_opmode_streaming)
vrep.simxSetJointTargetVelocity(robot[0], robot[3],
 velocity * .03, vrep.simx_opmode_streaming)
if flag:
    while abs(angle_degrees) > abs(angle_objective - 2):
        _, angle = vrep.simxGetObjectOrientation(robot[0],
 robot[1], -1, vrep.simx_opmode_streaming)
        angle_degrees = math.degrees(angle[2])
else:
    while angle_degrees < (angle_objective - 2):
        _, angle = vrep.simxGetObjectOrientation(robot[0],
 robot[1], -1, vrep.simx_opmode_streaming)
        angle_degrees = math.degrees(angle[2])

vrep.simxSetJointTargetVelocity(robot[0], robot[2], 0,
 vrep.simx_opmode_streaming)
vrep.simxSetJointTargetVelocity(robot[0], robot[3], 0,
vrep.simx_opmode_streaming)
```

```
def mov_robot_1(robot):
    start_time_1 = time.time()
    pos_1 = movement_forward(robot, 0.0)
    # Vertex 0
    robot1_path.append([pos_1[0], pos_1[1],
    time.time() - start_time_1])

    movement_right(robot)
    pos_1 = movement_forward(robot, 1.0)
    # Vertex 1
    robot1_path.append([pos_1[0], pos_1[1],
    time.time() - start_time_1])

    movement_left(robot)
    movement_left(robot)
    pos_1 = movement_forward(robot, 1.0)
    movement_right(robot)
    pos_1 = movement_forward(robot, 2.4)
    # Vertex 2
    robot1_path.append([pos_1[0], pos_1[1],
     time.time() - start_time_1])
    movement_right(robot)
    pos_1 = movement_forward(robot, 0.5)
    # Vertex 3
    robot1_path.append([pos_1[0], pos_1[1],
     time.time() - start_time_1])
    movement_right(robot)
    pos_1 = movement_forward(robot, 1.9)
    # Vertex 4
    robot1_path.append([pos_1[0], pos_1[1],
     time.time() - start_time_1])
    movement_left(robot)
    pos_1 = movement_forward(robot, 1.5)
    # Vertex 5
    robot1_path.append([pos_1[0], pos_1[1],
     time.time() - start_time_1])
    movement_left(robot)
```

```
pos_1 = movement_forward(robot, 0.9)
# Vertex 6
robot1_path.append([pos_1[0], pos_1[1],
 time.time() - start_time_1])
movement_right(robot)
movement_right(robot)
pos_1 = movement_forward(robot, 1.4)
# Vertex 7
robot1_path.append([pos_1[0], pos_1[1],
 time.time() - start_time_1])
movement_right(robot)
movement_right(robot)
pos_1 = movement_forward(robot, 0.5)
movement_left(robot)
pos_1 = movement_forward(robot, 1.5)
movement_right(robot)
pos_1 = movement_forward(robot, 1.9)
movement_right(robot)
pos_1 = movement_forward(robot, 0.5)
# Vertex 8
robot1_path.append([pos_1[0], pos_1[1],
 time.time() - start_time_1])
pos_1 = movement_forward(robot, 0.9)
# Vertex 9
robot1_path.append([pos_1[0], pos_1[1],
 time.time() - start_time_1])
movement_right(robot)
pos_1 = movement_forward(robot, 0.4)
# Vertex 10
robot1_path.append([pos_1[0], pos_1[1],
 time.time() - start_time_1])
movement_right(robot)
pos_1 = movement_forward(robot, 0.9)
# Vertex 11
robot1_path.append([pos_1[0], pos_1[1],
 time.time() - start_time_1])
movement_left(robot)
movement_left(robot)
pos_1 = movement_forward(robot, 0.85)
```

```
# Return to vertex 10
movement_left(robot)
pos_1 = movement_forward(robot, 0.4)
# Return to vertex 9
movement_left(robot)
pos_1 = movement_forward(robot, 0.9)
# Return to vertex 8
movement_right(robot)
pos_1 = movement_forward(robot, 0.4)
# Return to the origin of the third zone
# Vertex 12
robot1_path.append([pos_1[0], pos_1[1],
 time.time() - start_time_1])
pos_1 = movement_forward(robot, 0.35)
#vertex 13
robot1_path.append([pos_1[0], pos_1[1],
 time.time() - start_time_1])
pos_1 = movement_forward(robot, 1.15)
# vertex 14
robot1_path.append([pos_1[0], pos_1[1],
 time.time() - start_time_1])


movement_right(robot)
movement_right(robot)
pos_1 = movement_forward(robot, 0.9)
# Return to vertex 13
movement_left(robot)
# Starts the fourth zone
pos_1 = movement_forward(robot, 1.0)
# vertex 15
robot1_path.append([pos_1[0], pos_1[1],
 time.time() - start_time_1])
movement_left(robot)
# vertex 16
pos_1 = movement_forward(robot, 0.9)
robot1_path.append([pos_1[0], pos_1[1],
 time.time() - start_time_1])
movement_right(robot)
```

```
movement_right(robot)
pos_1 = movement_forward(robot, 0.85)
# Return to vertex 15
movement_right(robot)
pos_1 = movement_forward(robot, 0.9)
# Return to vertex 13
movement_left(robot)
# Return to the origin of the fourth zone
pos_1 = movement_forward(robot, 0.5)
movement_right(robot)
pos_1 = movement_forward(robot, 0.5)
# vertex 17
robot1_path.append([pos_1[0], pos_1[1],
time.time() - start_time_1])

# Start the fifth zone
movement_right(robot)
pos_1 = movement_forward(robot, 1.45)
# vertex 18
robot1_path.append([pos_1[0], pos_1[1],
 time.time() - start_time_1])
movement_left(robot)
pos_1 = movement_forward(robot, 0.4)
# vertex 19
robot1_path.append([pos_1[0], pos_1[1],
 time.time() - start_time_1])

# Return to the origin of the fourth zone
movement_right(robot)
movement_right(robot)
pos_1 = movement_forward(robot, 0.4)
# Return to vertex 18
movement_right(robot)
pos_1 = movement_forward(robot, 1.45)
# Return to vertex 17
movement_left(robot)
pos_1 = movement_forward(robot, 0.5)
movement_right(robot)
pos_1 = movement_forward(robot, 0.4)
```

```
    movement_left(robot)
    pos_1 = movement_forward(robot, 0.8)
    # Return to vertex 9
    # Don't erase it because it saves the time
    robot1_path.append([pos_1[0], pos_1[1],
    time.time() - start_time_1])


def mov_robot_2(robot):
    start_time_2 = time.time()
    pos_2 = movement_forward(robot, 0.0)
    # Vertex 0
    robot2_path.append([pos_2[0], pos_2[1],
    time.time() - start_time_2])

    pos_2 = movement_forward(robot, 0.5)
    # Vertex 1
    robot2_path.append([pos_2[0], pos_2[1],
    time.time() - start_time_2])
    pos_2 = movement_forward(robot, 0.5)
    # Vertex 2
    robot2_path.append([pos_2[0], pos_2[1],
    time.time() - start_time_2])
    movement_left(robot)
    pos_2 = movement_forward(robot, 0.5)
    # Vertex 3
    robot2_path.append([pos_2[0], pos_2[1],
    time.time() - start_time_2])
    movement_right(robot)
    pos_2 = movement_forward(robot, 1.0)
    # Vertex 4
    robot2_path.append([pos_2[0], pos_2[1],
    time.time() - start_time_2])
    movement_right(robot)
    pos_2 = movement_forward(robot, 0.5)
    # Vertex 5
    robot2_path.append([pos_2[0], pos_2[1],
    time.time() - start_time_2])
    movement_left(robot)
```

```
movement_left(robot)
pos_2 = movement_forward(robot, 1.85)
# Vertex 6
robot2_path.append([pos_2[0], pos_2[1],
time.time() - start_time_2])
movement_left(robot)
movement_left(robot)
pos_2 = movement_forward(robot, 1.3)
robot2_path.append([pos_2[0], pos_2[1],
time.time() - start_time_2])
movement_right(robot)
movement_left(robot)
movement_left(robot)
pos_2 = movement_forward(robot, 0.4)
# Vertex 8
robot2_path.append([pos_2[0], pos_2[1],
time.time() - start_time_2])
movement_right(robot)
pos_2 = movement_forward(robot, 0.5)
# Vertex 9
robot2_path.append([pos_2[0], pos_2[1],
time.time() - start_time_2])
movement_right(robot)
pos_2 = movement_forward(robot, 1.00)
# Vertex 10
robot2_path.append([pos_2[0], pos_2[1],
time.time() - start_time_2])
pos_2 = movement_forward(robot, 0.85)
# Vertex 11
robot2_path.append([pos_2[0], pos_2[1],
time.time() - start_time_2])
movement_left(robot)
movement_left(robot)
pos_2 = movement_forward(robot, 0.8)
# Return to vertex 10
movement_right(robot)
pos_2 = movement_forward(robot, 0.5)
# Vertex 12
robot2_path.append([pos_2[0], pos_2[1],
```

```
    time.time() - start_time_2])
    # this part is for the fourth zone
    movement_right(robot)
    pos_2 = movement_forward(robot, 0.4)
    # Vertex 13
    robot2_path.append([pos_2[0], pos_2[1],
    time.time() - start_time_2])
    movement_left(robot)
    pos_2 = movement_forward(robot, 0.5)
    # Vertex 14
    robot2_path.append([pos_2[0], pos_2[1],
    time.time() - start_time_2])
    movement_right(robot)
    pos_2 = movement_forward(robot, 0.43)
    # Vertex 15
    robot2_path.append([pos_2[0], pos_2[1],
    time.time() - start_time_2])
    movement_left(robot)
    pos_2 = movement_forward(robot, 1.0)
    # Vertex 16
    robot2_path.append([pos_2[0], pos_2[1],
     time.time() - start_time_2])
    movement_right(robot)  # left
    movement_right(robot)
    pos_2 = movement_forward(robot, 0.9)
    # Return to vertex 15
    pos_2 = movement_forward(robot, 1.35)
    # Vertex 17
    robot2_path.append([pos_2[0], pos_2[1],
    time.time() - start_time_2])
    movement_right(robot)
    movement_left(robot)
    movement_left(robot)
    pos_2 = movement_forward(robot, 0.93)
    # Vertex 18
    robot2_path.append([pos_2[0], pos_2[1],
    time.time() - start_time_2])
    movement_left(robot)
    pos_2 = movement_forward(robot, 1.45)
```

```
    # Vertex 19
    robot2_path.append([pos_2[0], pos_2[1],
    time.time() - start_time_2])
    # return movement
    movement_right(robot)
    movement_right(robot)
    pos_2 = movement_forward(robot, 1.45)
    # Return to vertex 18
    robot2_path.append([pos_2[0], pos_2[1],
    time.time() - start_time_2])


def mov_robot_3(robot):
    robot3_path.append([0, 0, 0])
    robot3_path.append([0, 0, 0])
    pass


print ('Program started')
vrep.simxFinish(-1) s
clientID_R1 = vrep.simxStart('127.0.0.1', 19999,
True, True, 5000, 5)  # Connect to V-REP robot 1
clientID_R2 = vrep.simxStart('127.0.0.1', 20000,
True, True, 5000, 5)  # Connect to V-REP robot 1
clientID_R3 = vrep.simxStart('127.0.0.1', 20001,
True, True, 5000, 5)  # Connect to V-REP robot 1

w = 0.10  # Angular velocity .02 -> 2rad/s
w_factor = w / 3.
velocity = w_factor * 100.

if clientID_R2 != -1 and clientID_R2 != -1 and clientID_R3 != -1:
    print ('Connected to remote API server')

    _, left_motor_handle = vrep.simxGetObjectHandle(clientID_R1,
     'left_joint', vrep.simx_opmode_oneshot_wait)
    _, right_motor_handle = vrep.simxGetObjectHandle(clientID_R1,
     'right_joint', vrep.simx_opmode_oneshot_wait)
    _, sensorHandle = vrep.simxGetObjectHandle(clientID_R1,
```

```
 'sensor_collection', vrep.simx_opmode_oneshot_wait)
_, robot_1 = vrep.simxGetObjectHandle(clientID_R1, 'Robot_1',
 vrep.simx_opmode_oneshot_wait)

_, left_motor_handle_2 = vrep.simxGetObjectHandle(clientID_R2,
 'left_joint_2', vrep.simx_opmode_oneshot_wait)
_, right_motor_handle_2 = vrep.simxGetObjectHandle(clientID_R2,
 'right_joint_2', vrep.simx_opmode_oneshot_wait)
_, sensorHandle_2 = vrep.simxGetObjectHandle(clientID_R2,
'sensor_collection_2', vrep.simx_opmode_oneshot_wait)
_, robot_2 = vrep.simxGetObjectHandle(clientID_R2, 'Robot_2',
vrep.simx_opmode_oneshot_wait)

_, left_motor_handle_3 = vrep.simxGetObjectHandle(clientID_R3,
'left_joint_3', vrep.simx_opmode_oneshot_wait)
_, right_motor_handle_3 = vrep.simxGetObjectHandle(clientID_R3,
'right_joint_3', vrep.simx_opmode_oneshot_wait)
_, sensorHandle_3 = vrep.simxGetObjectHandle(clientID_R3,
'sensor_collection_3', vrep.simx_opmode_oneshot_wait)
_, robot_3 = vrep.simxGetObjectHandle(clientID_R3, 'Robot_3',
vrep.simx_opmode_oneshot_wait)

robot_11 = [clientID_R1, robot_1, left_motor_handle,
right_motor_handle]
robot_22 = [clientID_R2, robot_2, left_motor_handle_2,
right_motor_handle_2]
robot_33 = [clientID_R3, robot_3, left_motor_handle_3,
right_motor_handle_3]
i = 0
while i < 3:
    returnCode_1, position = vrep.simxGetObjectPosition(
    robot_11[0], robot_11[1], -1, vrep.simx_opmode_streaming)
    return_code_2, orientation = vrep.simxGetObjectOrientation(
    robot_11[0], robot_11[1], -1,
                                    vrep.simx_opmode_streaming)
    returnCode_1, position = vrep.simxGetObjectPosition(
robot_22[0], robot_22[1], -1, vrep.simx_opmode_streaming)
    return_code_2, orientation = vrep.simxGetObjectOrientation(
    robot_22[0], robot_22[1], -1,
```

```
        vrep.simx_opmode_streaming)
        returnCode_1, position = vrep.simxGetObjectPosition(
        robot_33[0], robot_33[1], -1, vrep.simx_opmode_streaming)
        return_code_2, orientation = vrep.simxGetObjectOrientation(
        robot_33[0], robot_33[1], -1,
        vrep.simx_opmode_streaming)
        time.sleep(0.5)
        i += 1
        print "."

r1 = threading.Thread(target=mov_robot_1, args=(robot_11,))
r2 = threading.Thread(target=mov_robot_2, args=(robot_22,))
r3 = threading.Thread(target=mov_robot_3, args=(robot_33,))

print "Movement starts"
r1.start()
r2.start()
r3.start()

r1.join()
r2.join()
r3.join()
print "Movement ends"

# print robot1_path
# print robot2_path
# print robot3_path

f = open("./data_r1", "w")
f.write("# x y\n")
np.savetxt(f, robot1_path)

f = open("./data_r2", "w")
f.write("# x y\n")
np.savetxt(f, robot2_path)

f = open("./data_r3", "w")
f.write("# x y\n")
np.savetxt(f, robot3_path)
```

```
    vrep.simxGetPingTime(clientID_R1)
    vrep.simxGetPingTime(clientID_R2)
    vrep.simxGetPingTime(clientID_R3)

    # Now close the connection to V-REP:
    vrep.simxFinish(clientID_R1)
    vrep.simxFinish(clientID_R2)
    vrep.simxFinish(clientID_R3)
else:
    print ('Failed connecting to remote API server')
print ('Program ended')
```