
Simulación de un algoritmo de control de colonias de robots móviles ruteadores para ambientes de exploración utilizando Ros y Gazebo



UNIVERSIDAD DEL BÍO-BÍO

Seminario de Título

Melquisedec Antonio Sierra Arcos

Profesores Guías:

Ph.D. Cristian Durán Faúndez, Ph.D. Ángel Rubio,

Ph.D. Fernando Von Zuben

Departamento de Ingeniería Eléctrica y Electrónica

Facultad de Ingeniería

Universidad del Bio Bio

Abril 2019

Simulación de un algoritmo de control de colonias de robots móviles ruteadores para ambientes de exploración utilizando Ros y Gazebo

*Seminario de Título que presenta para optar al título de Ingeniero
Civil en Automatización*

**Ingeniería Civil en Automatización
IT/2019/4**

Versión 1.0+

Profesores Guías:

Ph.D. Cristian Durán Faúndez, Ph.D. Ángel Rubio,

Ph.D. Fernando Von Zuben

Departamento de Ingeniería Eléctrica y Electrónica

Facultad de Ingeniería

Universidad del Bío Bío

Abril 2019

Copyright © Melquisedec Sierra Arcos

A mis padres, polola y amigos.

Agradecimientos

*La verdadera felicidad radica en la
finalización del trabajo utilizando tu
propio cerebro y habilidades.*

Soichiro Hona

Agradecimientos a mi familia, mi padre Romelio Sierra, mi madre Jeanette Arcos y polola Yarima Rabanl, por los retos de recordarme que debo terminar la tesis, por el apoyo de estar haciendo lo correcto y escuchar mis problemas, que muchas veces al realizar este acto pude encontrar soluciones a los mismos problemas.

A mis amigos de años Yonathan Guíñez, Carlos Beltran, Matias Contreras, mis amigos y familia brasilera de la república Panela que me han apoyado y subido el ánimo en varios momentos dandome a entender que soy capaz y que el esfuerzo valdrá la pena.

Al profesor Cristian Durán por aguantar mis lapsos de ausencia en todo el periodo de la tesis, que fue bastante, y por dedicarme bastante tiempo al final y ayudarme con el código implementado.

Al profesor Ángel Rubio por darme consejos de la estructura y cumplimiento de los objetivos de la tesis cuando no sabía que significaba realmente hacer una tesis.

Al profesor Fernando Von Zuben ¹ por darme la oportunidad de asistir a sus calses en la Universidad Estadual de Campinas, Brasil, ya que, me ayudó bastante a entender el funcionamiento de la Inteligencia Artificial . Finalmente a mis jefes en trabajos laborales que para bien o para mal me motivaron a terminar la tesis para ser un profesional con todas sus letras.

¹<http://www.dca.fee.unicamp.br/vonzuben/>

Resumen

...
...

Desde fines del siglo XX, apoyado por el creciente avance tecnológico, hasta la actualidad, la Robótica ha adquirido una relevancia importante a nivel mundial en todos los ámbitos como educación, investigación e industria. Es por esto que en el mundo de la Robótica poder llevar a cabo un algoritmo de Inteligencia Artificial específico puede significar una inversión muy alta, debido a esto es común utilizar softwares de simulación para reducir este costo al mínimo. Esta tesis busca dar una herramienta útil para probar distintos algoritmos de Inteligencia Artificial para una aplicación de exploración hipotética, utilizando como sistema operativo Ubuntu para programación más compatible, ROS a través de programación en C++ y Gazebo como software de simulación en tres dimensiones.

Índice

Agradecimientos	VII
Resumen	VIII
1. Introducción	1
1.1. Introducción	1
1.2. Alcances	2
1.3. Objetivos	2
1.3.1. Objetivos Específicos	2
1.4. Contenido	3
2. Conceptos Generales	4
2.1. Robótica	4
2.2. Robótica de enjambre	4
2.3. Inteligencia Artificial	5
2.3.1. Machine Learning	6
2.4. Simulación y Software	7
2.4.1. Sistemas Operativos	7
2.4.1.1. Windows	7
2.4.1.2. Linux	8
2.4.1.3. ROS	8
2.4.2. Gazebo, V-rep, Blender, Inventor, Meshlab	9
2.5. Conclusión parcial	10
2.5.1. Ubuntu	10
2.5.2. ROS	10
2.5.3. Gazebo	11
2.5.4. Machine Learning	11
3. Solución Propuesta	12
3.1. Caseo de Estudio	12
3.1.1. Problemática	12
3.1.2. Alcances	15

3.1.3.	Segmentación del Problema	15
3.1.3.1.	Q-Learning	16
3.1.3.2.	Algoritmos de programación	19
3.2.	Esquema General	20
3.2.1.	Estructura	21
3.2.2.	Diagrama de flujo	23
3.3.	Robots a utilizar	24
3.3.1.	Estación Base(REB)	24
3.3.2.	Robot Router Autónomo(RRA)	25
3.3.3.	Robot Explorador Autónomo(REA)	26
3.4.	Código comunicación	26
3.4.1.	Machine Learning	27
4.	Lenguajes y aplicaciones utilizadas	28
4.1.	Lenguaje de programación XML	28
4.1.1.	Ventajas	29
4.1.2.	Estructura de un documento XML	29
4.1.2.1.	Prólogo	29
4.1.2.2.	Cuerpo	30
4.1.2.3.	Elementos	30
4.1.2.4.	Atributos	30
4.1.2.5.	Entidades predefinidas	31
4.1.2.6.	Comentarios	31
4.1.3.	Sintaxis	31
4.2.	XML Robot Description Format(URDF)	32
4.2.1.	Link	33
4.2.1.1.	Atributos	34
4.2.1.2.	elementos	34
4.2.2.	Joint	37
4.2.2.1.	atributos	37
4.2.2.2.	elementos	38
4.2.2.3.	Archivos Xacro Macro	41
5.	Implementación	42
5.1.	Introducción	42
5.1.1.	Main	42
5.1.2.	Listener	42
5.1.3.	Comandos	42
5.2.	Estado de los Robots	43
5.2.1.	Load	43
5.2.2.	OnUpdate	43

5.2.3.	buscar	44
5.2.4.	pintar	44
5.2.5.	cargarUniones	44
5.3.	Cantidad de Robots Router Autónomos	44
5.3.1.	Adicionar RRA	45
5.3.2.	Variables Globales Auxiliares	46
5.3.3.	IARobot	48
5.3.4.	enviarRos	48
5.4.	Inteligencia Artificial	49
5.4.1.	calcularRSSI	49
5.4.2.	Accion	49
5.4.3.	getDeseado	51
5.5.	Trayectoria del Robot Explorador Autónomo	51
5.5.1.	procesar	51
5.5.2.	mover	52
5.5.3.	parametrizar	52
5.5.4.	Trayectoria zigzag	53
5.5.5.	Trayectoria L Derecha	53
5.6.	Iniciar Simulación	54
6.	Análisis de Resultados	55
6.1.	Introducción	55
6.2.	Recurso computacional	55
6.2.1.	Análisis Monitor de Sistema	56
6.2.2.	Información por consola	59
6.2.2.1.	Debug Gazebo	60
6.2.2.2.	Estado robots	61
6.3.	Visualización	61
6.3.1.	Análisis trayectorias	61
6.3.1.1.	Trayectoria zigzag	62
6.3.1.2.	Trayectoria L derecha	63
6.4.	Análisis de variables	64
6.4.0.1.	Trayectoria zigzag	64
6.4.0.2.	Trayectoria L derecha	66
7.	Conclusiones	68
7.1.	Conclusión	68
7.2.	Trabajos futuros	69
7.2.1.	Cambio de Robots	69
7.2.2.	Modo trayectoria	69
7.2.3.	Cambio de parámetros de los robots	69

7.2.4. Utilización de sensores simulados	69
7.2.5. Optimización del código	70
7.2.6. HMI con visualización de estados	70
I Anéxos	71
A. ROS	72
A.1. Introducción a ROS	72
A.2. Conceptos básicos	73
A.3. Componentes Principales	75
A.3.1. Infraestructura de comunicaciones	75
A.3.2. Características Robóticas específicas	75
A.3.2.1. Mensajes estándar de robot	75
A.3.2.2. Librería geométrica de robot	75
A.3.2.3. Unidades Utilizadas	76
A.3.2.4. Estructura del Robot	76
A.3.2.5. Diagnósticos	77
A.3.3. Herramientas	78
A.3.3.1. Rosnode	78
A.3.3.2. Rostopic	78
A.3.3.3. Rosservice	79
A.3.3.4. Rosparam	79
A.3.3.5. Package	79
A.3.3.6. Rosbash	80
A.3.4. Integración con librerías	80
B. Gazebo	81
B.1. Introducción a Gazebo	81
B.2. Componentes Principales	81
B.3. Ejecución	81
B.4. Utilidades	82
B.4.1. Control de cámara	82
B.4.2. Diseño de Mundos	83
B.4.3. Visualización de Sensores	83
B.4.4. Importación de modelos 3D al mundo	84
B.4.5. Comandos Útiles	85
B.4.5.1. Movimiento de un robot	85

Bibliografía	87
---------------------	-----------

Lista de acrónimos	89
---------------------------	-----------

Índice de figuras

3.1. Modelo de solución de conectividad mediante Router Estático.	13
3.2. Modelo de solución de conectividad mediante Router Móvil. .	14
3.3. Esquema General Q-Learning.	20
3.4. Modelo final de estudio	20
3.5. Diagrama de Flujo general del programa de simulación. . . .	23
3.6. TurtleBot.	24
3.7. Esquema de movimiento del Robot Router Autónomo.	25
3.8. iRobot create.	25
3.9. Pioneer 2DX.	26
3.10. Esquema General de Comunicación	26
3.11. Esquema de funcionamiento de la Inteligencia Artificial. . . .	27
4.1. Ejemplo de estructura robótica.	33
4.2. Elemento Link	34
4.3. Elemento Joint	37
5.1. Estructura de un archivo *.world.	45
5.2. Estructura de model de un robot.	46
6.1. Información de Monitor de Sistema con programa detenido. .	56
6.2. Información de Monitor de Sistema con programa iniciado. .	57
6.3. Información de Monitor de Sistema con programa con 1 Robot Router Autónomo	58
6.4. Información de Monitor de Sistema con programa con 2 Robot Router Autónomo	59
6.5. Información mostrada a través de debug por consola.	60
6.6. Información mostrada por consola a través de un topic. . . .	61
6.7. Ejecución trayectoria zigzag	62
6.8. Ejecución trayectoria L derecha	63
6.9. Posición Robot Explorador Autónomo en trayectoria zigzag .	64
6.10. Difa, RSSIsup y RSSIinf de cada RRA en la trayectoria zigzag.	65
6.11. Posición Robot Explorador Autónomo en trayectoria L derecha	66

6.12. Difa, RSSIsup y RSSIinf de cada RRA en la trayectoria L derecha.	67
A.1. Diagrama de comunicación Nodo a Nodo	73
A.2. Geometría de Robots en ROS	76
B.1. Vista del mundo simulado por gazebo con un robot turtlebot	82
B.2. Visualizacion en Rviz de la obtencion de datos del robot Turtle- bot	83
B.3. Visualizacion del mundo creado con dos robots	84

Índice de Tablas

5.1. Cumplimiento de objetivos	50
5.2. Accion deseada v/s Orientacion robot	51
5.3. Contenido Archivo zigzag	53
5.4. Contenido Archivo diader	54
A.1. Unidades Básicas	76
A.2. Unidades Derivadas	77

Capítulo 1

Introducción

El primer paso es establecer que algo es posible; entonces la probabilidad ocurrirá.

Elon Musk

1.1. Introducción

Para este Trabajo de Título se analizará un caso de un problema de control de colonias de robots ruteadores, en aplicaciones de exploración, se puede reducir a lo siguiente: Se cuenta con un Robot Autónomo Explorador que se desplaza de forma libre por un ambiente, comenzando en una estación base con la que se debe mantener un lazo de comunicación inalámbrica. El enlace de comunicación se ve afectado por diversos factores, donde uno de los principales está asociado a la distancia entre el explorador y la base. Para mantener ciertos niveles de calidad de enlace, robots ruteadores son desplegados entre el explorador y la base, en una cadena, disminuyendo la distancia entre cada salto inalámbrico. El objetivo del control de la colonia de robots routers es llevar a cada robot router a una coordenada tal que la calidad del enlace sea (idealmente) maximizada Olate y Duran-Faundez (6) . Para el estudio de soluciones para problemas de este tipo, herramientas de simulación son necesarias. En (Olate y Duran-Faundez (6)), un algoritmo de decisiones basado en machine learning es propuesto y simulado en software Matlab. Por otra parte, el estado del arte propone herramientas específicas para robótica, y la utilización de software genérico como Matlab es cuestionada por la comunidad del área, pues obliga a hacer simplificaciones demasiado importantes. En robótica, uno de los software más utilizados para la programación de robots, y que goza de aceptación por parte de la comunidad internacional, es ROS¹, el cual es utilizado por importantes empresas y universidades como

¹Robotics Operation System

por ejemplo: Instituto Tecnológico de Aragon², Asimov Robotic³, Universidad de León⁴, Universidad Carlos III de Madrid⁵, ARIAC⁶, SRI International⁷, NVIDIA⁸, Pioneer⁹. Junto con este, el software GAZEBO¹⁰ permite la simulación de robots utilizando ROS en ambientes virtuales, permitiendo la interacción con objetos y obstáculos. Este Trabajo de Título tiene por objetivo generar una herramienta computacional en Gazebo capaz de simular y demostrar visualmente el comportamiento, manipulado por un algoritmo de la literatura, de múltiples robots móviles ruteadores en un ambiente virtual ejemplificador utilizando ROS como sistema base.

1.2. Alcances

Para efectos de este Trabajo de Título sólo se utilizará un algoritmo de Inteligencia Artificial para probar que la herramienta computacional creada funciona correctamente. El algoritmo utilizado será abordado de manera global, ya que, el objetivo de este trabajo es crear la herramienta computacional capaz de simular correctamente el algoritmo en un ambiente virtual. Además el ambiente virtual creado no tendrá obstáculos y los robots serán los que contiene el simulador Gazebo en su librería.

1.3. Objetivos

El Objetivo Principal de este Trabajo de Título es generar una herramienta computacional capaz de simular el comportamiento de una colonia de Robots Autónomos Exploradores, mediante la utilización de ROS y el simulador GAZEBO para un problema de exploración de un ambiente desconocido con robots.

1.3.1. Objetivos Específicos

1. Generar un programa en ROS de fácil manipulación para ser simulado en GAZEBO.
2. Implementar un algoritmo de la literatura para la resolución de un problema de mantención de enlaces de comunicación inalámbricos en colonia de Robots Autónomos Exploradores .

²<http://www.itainnova.es>

³<http://www.asimovrobotics.com/>

⁴<http://www.unileon.es/>

⁵<http://www.uc3m.es/Inicio>

⁶<http://gazebosim.org/ariac>

⁷<https://www.sri.com/>

⁸<http://robots.ros.org/panther/>

⁹<http://robots.ros.org/pioneer-3-at/>

¹⁰<http://gazebosim.org/>

3. Crear un ambiente virtual, utilizando GAZEBO, que permita visualizar el funcionamiento del algoritmo a utilizar.
4. Analizar el algoritmo seleccionado utilizando la herramienta desarrollada para el problema de mantención de enlaces de comunicación inalámbricos en colonia de Robots Autónomos Exploradores .

1.4. Contenido

Para tener un orden general y específico del documento elaborado se dividieron 7 capítulos.

El Capítulo 1 aborda el motivo del desarrollo del seminario de título, sus objetivos generales y específicos y los alcances del proyecto.

El Capítulo 2 se aborda el estado del arte, en el cual se mencionará los distintos software, conceptos de robotica, robótica de enjambre, inteligencia artificial, deepLearning y sistemas operativos.

El Capítulo 3 se analiza la solución propuesta definiendo el caso de estudio, Q-Learning, esquema general y específico del comportamiento de los robots.

El Capítulo 4 se da a conocer el lenguaje XML utilizado por gazebo para la creación de robots.

El Capítulo 5 contiene la descripción global de las funciones utilizadas para el correcto funcionamiento del programa creado.

El Capítulo 6 se analiza el funcionamiento y rendimiento del programa de simulación propuesto.

El Capítulo 7 contiene la conclusión final del proyecto y algunas ideas de seguimiento y mejora del programa creado.

Capítulo 2

Conceptos Generales

*Grandes descubrimientos y mejoras
implican invariablemente la cooperación
de muchas mentes.*

Alexander Graham Bell.

2.1. Robótica

La robótica es un concepto de dominio público y la mayor parte de la gente tiene una idea de lo que és, sabe sus aplicaciones y el potencial que tienen. El desarrollo de tecnología, donde se incluyen las poderosas computadoras electrónicas, los actuadores de control retroalimentados, transmisión de potencia a través de engranes, y la tecnología en sensores, han contribuido al desarrollo amplio de la robótica en ámbitos industriales, domiciliarios y agricultura (Molina (5)).

2.2. Robótica de enjambre

La robótica de enjambre es una aproximación a la coordinación de un alto número de robots relativamente simples. De manera que en conjunto lleven a cabo tareas colectivas que sobrepasan las capacidades de un único robot (Dorigo et al. (3)). Utilizar este comportamiento no es casual, es inspirado en el estudio de la manera de actuar de insectos sociales, como hormigas, termitas, avispas o abejas, que son un gran ejemplo de colaboración de individuos simples que interactúan para crear sistemas inteligentes colectivos. Estos conjuntos de insectos demuestran tres características importantes para la robótica: robustez, flexibilidad y escalabilidad. Por lo tanto, la robótica de enjambre se basa en la metáfora de las colonias de insectos sociales para enfatizar aspectos como el control descentralizado, la comunicación limitada entre agentes, el uso de información local, la aparición de un comportamiento

global y la robustez (Dorigo et al. (4)). En (Şahin (9)) se propone la siguiente definición para este término: "La robótica de enjambre es el estudio de cómo un gran número de agentes relativamente simples pueden ser diseñados de manera que el comportamiento colectivo deseado emerja a partir de las interacciones locales entre los agentes y entre los agentes y el entorno". Para el caso particular de exploración de un ambiente desconocido (Zonas de desastres naturales, misiones militares, exploración planetaria, etc), una forma de explorarlo es mediante el uso de un enjambre de Robots Autónomos Exploradores, estos, generalmente, se encuentran con varios obstáculos de distinta índole, pero uno de ellos considerado importante es el mantener comunicación confiable con la estación base, esto es debido a la falta de repetidores en el área inexplorada y limitado al alcance de la señal de los mismos robots, la estación base o por la geografía que impiden la propagación de la señal.

2.3. Inteligencia Artificial

Desde hace un poco más de medio siglo nació una ciencia denominada "Inteligencia Artificial", la cual busca simular el proceso cognitivo que realiza la naturaleza, persona o ser vivo para la resolución de problemas o, dicho de una manera más formal, "Conjunto de técnicas que se aplican a la computadora con el fin de imitar el comportamiento humano". Sus objetivos principales son:

- Realizar investigaciones sobre el comportamiento humano para desarrollar herramientas de apoyo que ayuden a mejorar las actividades del hombre.
- Desarrollar y aplicar técnicas para representar un gran volumen de conocimientos, para resolver problemas complejos de una manera rápida y confiable.
- Lograr "crear pensamiento" y poder desarrollar sistemas diferentes a los convencionales que no toman decisiones propias.

A través de los años esta ciencias se fue dividiendo en varias ramas, tales como:

- Algoritmos genéticos.
- Redes Neuronales.
- Support Vector Machine
- Lógica Difusa.
- Machine Learning o Aprendizaje Automático.
- DeepLearning.

2.3.1. Machine Learning

Machine Learning o Aprendizaje automático es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden autónomamente. Todos los seres vivos exhiben algún tipo de comportamiento, en el sentido que realizan alguna acción como respuesta a un estímulo percibido por el ambiente en que se desenvuelven. Algunos de ellos, además, modifican su comportamiento a lo largo del tiempo, de forma que ante estímulos equivalentes se comportan de forma distinta con el paso del tiempo, es por esto que decimos que estos seres vivos han aprendido un comportamiento adecuado para interactuar con su entorno en su beneficio. Esto se puede ejemplificar en la enseñanza de un perro para hacer algún truco que nosotros deseamos. Para un mayor entendimiento de este tema se dividió en tres aéreas:

- Aprendizaje Supervisado: Técnica para deducir una función a partir de datos de entrenamiento catalogados en correctos e incorrectos.
- Aprendizaje No Supervisado: Modelo ajustado a las observaciones, es decir no se tiene conocimiento a priori de los datos de entrenamiento.
- Aprendizaje por Refuerzo: Algoritmo de aprendizaje que recibe algún tipo de valoración acerca de la idoneidad de la respuesta dada. Es similar al aprendizaje supervisado, en lo referente a recibir información acerca de lo que es apropiado, pero la gran diferencia es su enfrentamiento con las respuestas erróneas, cuando el aprendiz responde de forma inadecuada el aprendizaje supervisado le dice exactamente qué hacer en ese caso, por otro lado, el aprendizaje por refuerzo solo le informa acerca de que el comportamiento ha sido inapropiado y cuanto error se ha cometido.

En (Caparrini (1)) se habla sobre el proceso de simular el aprendizaje de sistemas biológicos reales y algunas suposiciones que simplifican el comportamiento de los agentes (aprendices). Esto permitirá tener un sistema flexible para proyectar diversas situaciones con el mismo sistema, y así poder extraer conclusiones generales acerca de las propiedades de los algoritmos que implementen estos sistemas de aprendizaje. En general se debe suponer que los aprendices siguen en su decisión un Proceso de Decisión de Markov, es decir:

- El agente percibe un conjunto finito, S , de estados distintos en su entorno, y dispone de un conjunto finito, A , de acciones para interactuar con él.
- El tiempo avanza de forma discreta, y en cada instante de tiempo, t , el agente percibe un estado concreto, $s_{(t)}$, selecciona una acción posible, $a_{(t)}$, y la ejecuta, obteniendo un nuevo estado, $s_{t+1} = a_t(s_t)$.

- El entorno responde a la acción del agente por medio de una recompensa, o castigo, $r(s_t, a_t)$ el cual se representara por un número, de forma que cuando mayor es, mayor es el beneficio.
- Tanto la recompensa como el estado siguiente obtenido no tienen por qué ser conocidos a priori por el agente y dependen únicamente del estado actual y de la acción tomada. Es decir, antes de aprender, el agente no sabe qué pasará cuando toma una acción determinada en un estado particular. Precisamente, un buen aprendizaje es aquel que permite adelantarse al agente en las consecuencias de las acciones tomadas, reconociendo las acciones que sobre estados concretos le llevan a conseguir con más eficacia y mayores recompensas, sus objetivos.

2.4. Simulación y Software

Hoy en día cuando se habla de simulación de cualquier tipo de eventos se piensa en un computador, sin embargo existen varios software dedicados a simular lo que la imaginación permita y, para llevar a cabo esto, se necesita un sistema operativo capaz de soportar dicho software es por esto que se analizarán algunas de las posibles herramientas informáticas que disponibles en la actualidad.

2.4.1. Sistemas Operativos

Un sistema operativo es un programa que controla otras partes del ordenador tanto hardware como software y permite además al usuario acceder a las facilidades que ofrece el sistema. Es por esto que para poder desarrollar una aplicación computacional de robótica se requiere algunos requisitos específicos para lograr un buen resultado, debido a esto que la elección del sistema operativo a utilizar tanto para programar como para ejecutar es de suma importancia.

2.4.1.1. Windows

Es un sistema operativo mundialmente reconocido y utilizado por usuarios y profesionales de las distintas áreas, sin embargo a nivel de desarrollador tiene algunas limitancias en relación a los permisos a los cuales se puede acceder. Por otra parte tiene una gran variedad de softwares que se pueden utilizar y el mercado cibernético está echo para cubrir cualquier necesidad a cambio de dinero. Windows es un sistema operativo desarrollado por la empresa de software Microsoft Corporation, el cual se encuentra dotado de una interfaz gráfica de usuario basada en el prototipo de ventanas (su nombre en inglés). Una ventana representa una tarea ejecutada o en ejecución,

cada una puede contener su propio menú u otros controles, y el usuario puede ampliarla o reducirla mediante un dispositivo señalador como el ratón o mouse.

2.4.1.2. Linux

Linux es una familia de sistemas operativos de tipo Unix que utilizan el kernel Linux. Es multiplataforma desde teléfonos móviles hasta supercomputadores. Se compone de un gran número de piezas que son desarrolladas de forma independiente por miles de programadores y proyectos. Normalmente estas piezas son integradas por un distribuidor y Linux es suministrado como una distribución Linux. Su licencia GPL ¹ y actualmente hay mas de distribuciones Linux de las cuales aproximadamente la mitad tienen un desarrollo activo. Estas distribuciones están adaptadas para usuarios o tareas específicas. Algunas de estas distribuciones están desarrolladas o apoyadas por empresas como Fedora ², openSUSE ³ y Ubuntu ⁴, mientras que otras son mantenidas por la propia comunidad de usuarios ⁵. Algunas de las distribuciones más conocidas son:

- Ubuntu
- Kubuntu
- Linux mint
- Debian

2.4.1.3. ROS

Este middleware (buscar sinonimo español), cuyas siglas derivan de la nomenclatura anglosajona Robot Operating System", tiene como propósito aportar un marco de trabajo común sobre el que articular una plataforma para robótica. De analizar sus principales características, se extrae que ROS puede aportar un gran número de ventajas:

- abstracción del hardware.
- gestión de las comunicaciones de forma transparente al usuario.
- gran número de herramientas(simuladores, Visualizadores de datos, depuradores,etc).

¹General Public License

²Red Hat

³Novel

⁴Canonical

⁵Debian

- librerías para robótica(matemáticas, geométricas, integración con visión por computador,etc).

Por último, señalar que existe una versión Industrial que se comenzo a trabajar desde el 2013.

2.4.2. Gazebo, V-rep, Blender, Inventor, Meshlab

- **Gazebo:** Software de simulación Open-Source, que ofrece la posibilidad de simular con precisión y eficiencia una grna cantidad de robots en entornos complejos. Incluye una funcionalidad en el motor de física que permite simular, entre otros, entornos acuáticos en los que los robots son influidos por corrientes y demás propiedades fluidodinámicas. Una característica relevante de este simulador es que tiene la capacidad de otorgar al desarrollador añadir plugins de físicas que afecten a una parte focalizada del entorno simulado, como puede ser un robot modificando su comportamiento ante ciertos eventos (a modo de ejemplo, podría recrearse un mundo con gravedad en el que un robot no esté afectado por ella).
- **V-rep:** Software con licencia gratuita y una comercial este simulador de robots tiene un entorno de desarrollo integrado basado en una arquitectura de control distribuido : cada objeto/modelo puede controlarse individualmente a través de un script incrustado, un complemento, un nodo ROS o Bluezero, un cliente API remoto o un sistema personalizado. Esto hace que sea muy versátil e ideal para aplicaciones multi-robot. Los controladores se pueden escribir en C/C++, Python, Java, Lua, Matlab u Octave.
- **Blender:** Software Open-Source destinado, en primera instancia, al modelado 3D de objetos. Incorpora la posibilidad de dar texturas, definir materiales, iluminar la escena, crear simulaciones. Destaca por sus capacidades para la creación de mallas y animaciones. Más aún, despunta al permitir que la comunidad de usuarios amplíe sus funcionalidades y características mediante scripts y plugins, desarrolladores principalmente en Python. En segundo término, Blender maneja una gran cantidad de formatos de entrada y salida, entre los que se encuentra el conocido DAE o COLLADA con el que se definen las geometrías de Gazebo. Por último, mencionar que permite generar, simplificar, combinar y modificar Point Clouds de forma relativamente sencilla.
- **Inventor:** Software privado que proporciona un conjunto de herramientas profesionales para diseño mecánico, documentación y simulación de productos en 3D. Al igual que toda la familia de productos AutoDesk

dispone de licencias educacionales fáciles de adquirir por estudiantes y entidades dedicadas a la enseñanza.

- **Meshlab:** Programa avanzado para trabajar con mallas tridimensionales, ampliamente conocido en el campo técnico del desarrollo 3D y manejo de datos. Esta herramienta destaca por su capacidad multiplataforma y su carácter Open Source, amparado bajo la licencia GNU (General Public License). En contraste con Blender, Meshlab entrega una gran cantidad de información estructural acerca de la malla cargada: número de vértices y aristas, conjunto de componentes normales a superficies, vértices... Además, posee una gran cantidad de herramientas y filtros con los que simplificar las mallas, reconstruir superficies, modificar el número de puntos que las conforman, sin alterar su estructura, y un largo etcétera.

2.5. Conclusión parcial

En esta sección se dará a conocer las explicaciones técnicas por las cuales se escogieron los Sistemas Operativos y Software utilizados en este Trabajo de Título.

2.5.1. Ubuntu

Como sistema Operativo principal de programación y ejecución del código se utilizará la distribución Ubuntu(basado en Debian) debido a la compatibilidad que ofrece para los softwares de código abierto, ya que, es una de las más utilizadas y se actualiza constantemente. En resumen a través de dos motivos:

- Comodidad para la programación e integración de proyectos de investigación aplicada.
- ROS se articula y construye en torno al núcleo de Ubuntu.

2.5.2. ROS

Debido a que la aplicación es específicamente para Robots y para facilitar el manejo en el mismo agente se utilizará ROS a través del lenguaje de programación c++ debido a su exactitud a bajo nivel en comparación con python que es de alto nivel. Cabe mencionar que gran parte de la Comunidad Investigadora acepta este SO. Además es escalable a la industria con su versión ROS-Industrial⁶.

⁶<https://rosindustrial.org/>

2.5.3. Gazebo

Se utilizará este software libre debido a su nulo gasto de inversión(en comparación a los otros mencionados) y la compatibilidad excelente con ROS y Ubuntu, ademas tiene las siguientes cualidades:

- Todas sus funcionalidades y características se centran y dedican únicamente al sector de la robótica permitiendo desde la sintonización de controladores hasta la detección de colisiones.
- Es multiplataforma, pudiendo ser utilizados por otros investigadores que prefieren utilizar otros sistemas operativos.
- Dispone de un motor de físicas renovado con el que simular incluso comportamientos fluido-mecánicos. En este aspecto, es sencillo encontrar simulaciones con drones(insertar referencia, submarinos(insertar referencia) e incluso estrategias de despegue de aviones(insertar referencia).
- Contempla la comunicación con un amplio espectro de interfaces como:
 - Matlab
 - Octave
 - C++
 - C#
 - Programas de realimentación de fuerzas para telemanipulación como Open Haptix (Insertar referencia).

Tanto es así, que ofrece la posibilidad de crear la simulación como si de un servidor se tratase y conectarlo al IoT("internet of Things")(insertar referencia) mediante esquemas basados en Javascript, como NodeJs y WebGL.(Insertar referencias)

2.5.4. Machine Learning

Como la aplicación para la cual se utilizará Inteligencia Artificial es de caracter de exploración, es decir, no se conoce el lugar, se debe utilizar un algoritmo en el cual el agente(robot) esté en constante aprendizaje, y específicamente se utilizará un código probado por un estudiante de la Universidad del Bio Bio (Olate y Duran-Faundez (6)), por su simpleza y efectividad demostrada en Matlab.

Capítulo 3

Solución Propuesta

*La simplicidad es la máxima
sofisticación.*

Leonardo da Vinci.

3.1. Caseo de Estudio

En los sistemas digitales y tecnologías inalámbricas, se van abriendo nuevos escenarios de estudio, algunos muy lejos de sus fundamentos electrónicos como lo es la dependencia que presenta la sociedad a la conectividad de redes de Internet y/o temas más cercanos al Hardware, como la gestión inteligente y eficiente de energía o direccionamiento y enrutamiento de paquetes de datos. En estos último años la transformación de redes cableadas a redes inalámbricas va en aumento, debido a la flexibilidad de conexión. Estas terminologías entran en juego en aplicaciones donde factores geográficos, confort del usuario o económicos, impiden la utilización de redes cableadas, implicando las virtudes y premuras inherente a la utilización del espacio libre como canal. Ante la necesidad de estar vinculados en tiempo real, transfiriendo información en una gran gama de escenarios, es preciso estudiar y diseñar formas de asegurar el flujo de datos en los nuevos escenarios en las que las redes inalámbricas se desempeñarán.

3.1.1. Problemática

Cuando se habla de utilizar medios inalámbricos, una condición fundamental es que la comunicación sólo puede efectuarse si se encuentran los dispositivos dentro de su área de cobertura. Si se desea establecer una conexión entre dos dispositivos inalámbricos, básicamente se pueden encontrar los siguientes escenarios:

1. Comunicación directa: Nodo emisor y nodo receptor pertenecen al área

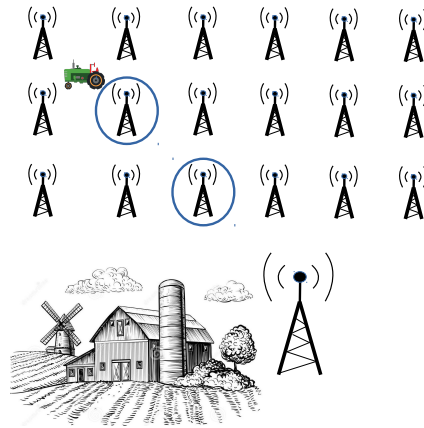


Figura 3.1: Modelo de solución de conectividad mediante Router Estático.

de cobertura del otro.

2. Comunicación indirecta: Caso cuando la distancia entre el nodo emisor y el nodo receptor superan el rango de cobertura, se pueden utilizar nodos intermedios (Gateway) que realicen enrutamientos de los paquetes de datos.

Cuando un usuario requiere comunicación continua con un terminal Tekdas y Terzis (10) exponen que, en una granja, considerando escenarios de trabajo fijos, la solución es la conectividad mediante un despliegue de router estáticos como se muestra en la Figura 3.1, si bien es factible y utilizada comúnmente, carece de factibilidad energética y económica, debido al alto consumo de electricidad, costo de instalación y mantención individual de cada nodo. Además cabe señalar que sólo se utiliza un pequeño conjunto de nodos para cumplir el objetivo como muestra la Figura 3.1 (antenas encerradas en círculo), mientras los demás no realizan acciones importantes en el enlace establecido, provocando consumo innecesario de energía. También la disposición de los nodos repetidores afectan la geografía del lugar.

Lo que Tekdas y Terzis (10) proponen es entregar conectividad mediante router móviles (ver Figura 3.2), que disminuye drásticamente la cantidad de nodos desplegados, comparado a la Figura 3.1. Teniendo en consideración la eficiencia energética y costos implicados, la opción más eficiente es la utilización de Robots Routers Autónomos, ya que, disminuye el número de unidades en funcionamiento y permite una organización dinámica de la red. En entornos donde se tendría poca o nula información de las características de la zona de trabajo, para mantener la conectividad entre unidades exploradoras de adquisición de datos y un punto de recopilación de información (base), Pezeshkian y Hart (7) exponen que un factor importante a

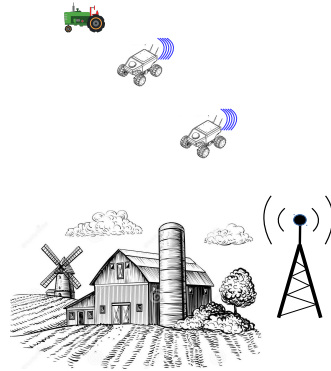


Figura 3.2: Modelo de solución de conectividad mediante Router Móvil.

considerar es la rápida degradación de la calidad de la comunicación al penetrar en el interior de edificios, túneles o cuevas. Para este problema ellos sugieren que una solución robusta es contar con un despliegue de unidades móviles, generando un convoy de robots que proporcionen redundancia de enlace a la unidad de interés. La característica de robustez en un enlace es deseable en casos donde la información es de vital importancia como: Operaciones militares terrestres, rescates, exploración y búsqueda. Todos estos escenarios se caracterizan por no poder contar con un conocimiento geográfico de la zona y requerir un despliegue dinámico de unidades.

Otra problemática a abordar es la confección de algoritmos de comportamiento para un sistema donde el enlace de datos es posible gracias a la utilización de nodos intermedios que proveen enrutamiento (Robot Router Autónomo), considerando además que su comportamiento no es aleatorio, a lo menos una de las unidades finales (Target o Gateway) son agentes móviles de comportamiento aleatorio. Con lo mencionado se puede inferir dos problemáticas:

1. El sistema de comunicación es una red inalámbrica móvil, lo que conlleva inconvenientes de pérdida de conexión o calidad de enlace debido a que la posición de los robots es dinámica.
2. El anclaje de red, proceso por el cual un dispositivo inalámbrico móvil actúa como pasarela para ofrecer conectividad de datos a otros dispositivos (Tethering), mediante nodos inalámbricos móviles equipados con algoritmos de control y tomando como parámetros de calidad de señal valores entregados por los dispositivos transceptores.

Se pretende dar un servicio de exploración o aseguramiento de terreno donde un robot explorador parte de un punto fijo o base a explorar una zona ale-

daña. Un ejemplo particular es en circunstancias donde se requiere explorar zonas de riesgo para las personas como estructuras colapsadas, debido a la irregularidad del terreno impiden una conexión simple, entran en juego la utilización de robots que actúan como Router Móviles asegurando la estabilidad del enlace entre la base y el robot explorador. Para solucionar este escenario, Olate y Duran-Faundez (6) proponen un algoritmo de Machine Learning denominado Q-Learning, cuya señal de entrada es el indicador de fuerza de la señal recibida (RSSI, por sus siglas en inglés) con un modelo log-distance Shadowing¹ con el cual se definen 3 indicadores:

1. Minimizar la diferencia absoluta (Min_{Dif_a}): Minimizar el módulo de la diferencia de RSSI de los vínculos prioritarios para lograr acercarse al punto óptimo (PO), que es parte de la recta de diferencia nula.
2. Maximizar el vínculo lejano (Max_{vl}): El objetivo a cumplir es función a un solo vínculo prioritario, en el cual el Router Robótico debe maximizar el valor de RSSI del vínculo prioritario más lejano.
3. Minimizar el vínculo cercano (Min_{vc}): Corresponde al opuesto de maximizar el vínculo lejano. Sin embargo, alejarse del vínculo cercano (minimizar el RSSI), no implica acercarse al vínculo lejano.

3.1.2. Alcances

Al desarrollar algoritmos de aseguramiento de enlace de comunicación inalámbrica mediante una colonia de robots, se presentan variadas restricciones que dependen del medio físico y del Hardware que se desea implementar. Debido a la infinita posibilidad de escenarios virtuales, este trabajo sólo abordará una temática ya estudiada en la misma casa de estudio (Olate y Duran-Faundez (6)).

3.1.3. Segmentación del Problema

En la problemática de brindar Aseguramiento de Conexión a un Dispositivo Mediante el Despliegue Dinámico de Robots Enrutadores, se plantea la siguiente segmentación del problema en función del análisis de las prestaciones del software de simulación y las consideraciones propuestas para el modelo de estudio. Se identifican tres sub-problemas:

1. Representación de una Estación Base de comunicación.
2. Comportamiento Individual de nodos de repetición móviles (mantención de vínculos).
3. Trayectoria del robot principal (tipo de instrucciones a seguir).

¹<https://www.gaussianwaves.com/2013/09/log-distance-path-loss-or-log-normal-shadowing-model/>

3.1.3.1. Q-Learning

Básicamente con este algoritmo el agente aprende a asignar valores de recompensa a los pares (estado, acción). Para continuar con el entendimiento de este algoritmo se necesita conocer las siguientes definiciones:

- Recompensa a largo plazo: premio que esperamos acumular a la larga si en cada estado realizamos la mejor acción posible.
- Recompensa directa: premio que se genera en una acción inmediata.
- Recompensa mixta: es una combinación de la recompensa a largo plazo con la recompensa directa, y se calcula de forma voraz (greedy).
- Experiencia: incluye un estado, la acción realizada, la recompensa recibida, y como novedad, el estado siguiente tras realizar esta acción.
- Velocidad de aprendizaje (learning rate): es un valor entre 0 y 1 que indica cuanto se puede aprender de cada experiencia:
 - 0: no aprende de la experiencia nueva.
 - 1: olvida todo lo que sabía hasta el momento y confía en la nueva experiencia completamente.
- Factor de Descuento (Discount Factor): es un valor entre 0 y 1 que indica cuán importante es el largo plazo:
 - 0: indica los refuerzos inmediatos son importantes.
 - 1: indica que los refuerzos inmediatos no son importantes, solo importa el largo plazo.
- Política: Objetivo de aprendizaje esperado.
- Problema MDP: cumple con los siguientes atributos:
 - S: conjunto de estados.
 - A: conjunto de acciones.
 - Estado inicial S_0 .
 - modelo de transiciones: $\sigma * (s, a, s')$.
 - Función de recompensa. $R(s)$.

Si un agente está en un determinado estado y toma una acción concreta, se debe observar la recompensa inmediata que recibe, pero también en las recompensas futuras que se obtendrán por pasar a otros estados donde se pueden tomar otras acciones, que seguirán una política particular. En el algoritmo Q learning, el valor Q de una par (estado, acción) contiene la

suma de todas estas posibles recompensas. El problema es que esta suma podría ser infinita en caso de que no haya un estado terminal que alcanzar. Además, es posible que no se quiera dar el mismo peso a las recompensas futuras, en cuyo caso se hace uso de un refuerzo acumulado con factor de descuento. El agente, al comienzo, no tiene información, por lo que su primer objetivo es aproximar lo mejor posible esta asignación de valores Q , como estos dependen de recompensas futuras y actuales, se proporciona un método capaz de calcular el valor final a partir de valores inmediatos y locales. Para ello:

- Si una acción en un estado determinado causa un resultado no deseado se debe aprender a no aplicar esa acción en ese estado. Si una acción en un estado determinado causa un resultado deseado se debe aprender a aplicar esa acción en ese estado.
- Si todas las acciones que se pueden tomar desde un estado determinado dan resultado negativo, es conveniente aprender a evitar ese estado. Es decir, no tomar acciones desde otros estados que nos lleven a él.
- Si cualquier acción en un determinado estado da un resultado positivo, se debe aprender que es conveniente llegar a él. Este hecho es lo que permite propagar la recompensa de par(estado, acción) a los pares de estados adyacentes.

Esto se modela matemáticamente de la siguiente manera:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_t + 1, a_t + 1)$$

Esto es, el valor de Q óptimo para un par (estado, acción) es la suma de la recompensa recibida cuando se aplica la acción junto al valor descontado del mejor valor Q que se puede conseguir desde el estado alcanzado al aplicar esa acción. Para aproximar este cálculo, al principio del aprendizaje, los valores Q se establecen a un valor fijo (puede ser aleatorio). A continuación el agente va tomando pares de (estado, acción) y anota cuánta recompensa recibe en ellos, entonces, actualiza el valor almacenado del valor Q de cada par considerando como ciertas las anotaciones tomadas de los otros pares (algunas de las cuales habrán sido aproximadas en pasos anteriores).

Una variante de la ecuación anterior podría ser la siguiente:

$$Q'(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha * [r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_t + 1, a_t + 1)]$$

Esta segunda ecuación intenta que la actualización de la función sea más gradual, no permitiendo cambios en una determinada dirección de forma tan brusca. Para ello, introduce un factor de aprendizaje, α , que controla la variación de Q . El nuevo valor de Q es la combinación ponderada del antiguo valor de Q y la nueva información que el agente debe creer. Un

aspecto importante del aprendizaje por refuerzo es la restricción de que solo se actualizan los valores Q de acciones que se ejecutan sobre estados por los que se pasa, no se aprende nada de acciones que no se intentan. Pero puede ser interesante que, sobre todo al principio del aprendizaje, el agente intente una gama más amplia de acciones, para hacerse una idea de lo que funciona y lo que no. Más concretamente, en cualquier momento el agente puede elegir:

- seleccionar una acción con el valor Q más alto para ese estado (explotación)
- seleccionar una acción al azar (exploración). Se formalizará este proceso de decisión en términos de probabilidades.

La posibilidad más simple es elegir completamente al azar entre una opción y otra, pero usaremos una opción un poco más inteligente y asignaremos una probabilidad a cada acción en función del valor Q asociado, de forma que cuanto mayor sea el valor Q de esa acción, mayor será la probabilidad de ser elegida. De esta forma, incluso las acciones con valores Q más bajos tendrán opciones de ser elegidas. También tiene sentido hacer que la probabilidad de exploración dependa del tiempo que el agente lleva aprendiendo, T , de tal forma que al principio se favorece la exploración porque lo que el agente haya aprendido todavía no es confiable y porque todavía quedan muchas opciones que no han sido consideradas y, más tarde se potencia la explotación, porque la experiencia del agente hace que su conocimiento sea más confiable. Así, la ecuación de la probabilidad de seleccionar la acción a_t sería:

$$P(a_t) = \frac{e^{E \cdot T \cdot Q(s_t, a_t)}}{\sum_{v_t} e^{E \cdot T \cdot Q(s_t, v_t)}}$$

Donde E es una constante de explotación, y v_t representa todas las posibles acciones que se pueden tomar desde s_t . Gracias a la función exponencial, a medida que T (temperatura) aumenta, las acciones que tengan un valor Q más alto se hacen mucho más grandes, por lo que tendrán más probabilidades de ser elegidas. En caso de que el espacio de estados sea una cantidad finita y manipulable, no es necesario realizar toda la segunda parte de selección de acciones para equilibrar la exploración con la explotación, basta realizar un recorrido completo de todos los posibles pares e ir actualizando el valor de Q . La convergencia del método asegura que obtendremos el resultado deseado.

3.1.3.2. Algoritmos de programación

En definitiva el algoritmo general de aprendizaje por refuerzo es:

```

for all  $(s, a)$  do
   $Q(s, a) \leftarrow \text{inicializar}$  //  $Q(a, b)$  es la estimación inicial de  $Q^*(s, a)$ 
end for
 $s \leftarrow \text{Observarestadoentorno}$ 
 $a \leftarrow \text{seleccionaraccin}$  // política de selección de acciones
while  $s! = \text{estado terminal}$  do
  ejecutar  $a$ 
   $r \leftarrow \text{reward}$ 
   $s' \leftarrow \text{nuevoestado}$ 
   $a' \leftarrow \text{seleccionarsiguienteaccin}$  // varias formas de actualizar  $Q(s, a)$ 
   $Q(s, a) \leftarrow \text{actualizar}$   $s \leftarrow s'$   $a \leftarrow a'$  end while

```

El Algoritmo Q-learning para entornos deterministas es:

```

for all  $(s, a)$  do
   $Q(s, a) \leftarrow \text{inicializar}$  //  $Q(a, b)$  es la estimación inicial de  $Q^*(s, a)$ 
end for
 $s \leftarrow \text{Observarestadoentorno}$ 
 $a \leftarrow \text{seleccionaraccin}$  // política de selección de acciones
while  $s! = \text{estado terminal}$  do
  ejecutar  $a$ 
   $r \leftarrow \text{reward}$ 
   $s' \leftarrow \text{nuevoestado}$ 
   $a' \leftarrow \text{seleccionarsiguienteaccin}$ 
   $Q(s, a) \leftarrow r + \gamma * \max_b Q(s', b)$ 
   $s \leftarrow s'$ 
   $a \leftarrow a'$ 
end while

```

El Algoritmo Q-learning para entornos No deterministas es:

```

for all  $(s, a)$  do
   $Q(s, a) \leftarrow \text{inicializar}$  //  $Q(s, a)$  es la estimación inicial de  $Q^*(s, a)$ 
end for
 $s \leftarrow \text{Observarestadoentorno}$ 
 $a \leftarrow \text{seleccionaraccin}$ 
// política de selección de acciones
while  $s! = \text{estado terminal}$  do
  ejecutar  $a$ 
   $r \leftarrow \text{reward}$ 
   $s' \leftarrow \text{nuevoestado}$ 

```

$a' \leftarrow \text{seleccionarsiguienteaccin}$
 $Q(s, a) \leftarrow Q(s, a) + \alpha * [r + \gamma * \max_b Q(s', b) - Q(s, a)]$
 $s \leftarrow s' \ // \ 0 < \alpha < 1 : \text{learning step}$
 $a \leftarrow a' \ \mathbf{end \ while}$

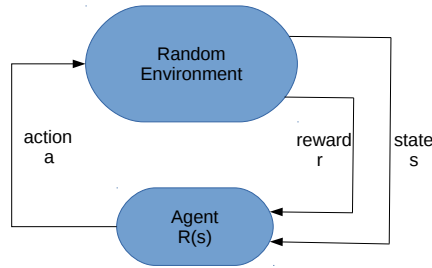


Figura 3.3: Esquema General Q-Learning.

Y en general, está demostrado en el esquema de la Figura 3.3

3.2. Esquema General

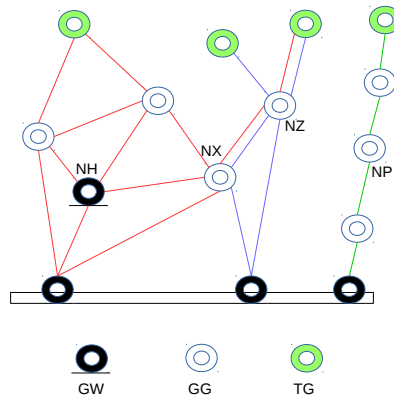


Figura 3.4: Modelo final de estudio

Por comportamiento general se entiende la inteligencia que planifica el funcionamiento del conjunto total de elementos que conforman el sistema

(colonia de robots). Comprende los algoritmos que permiten realizar tareas como: activación y desactivación de enlaces, determinación de vínculos prioritarios, interconexión entre diferentes link de comunicaciones. Todo esto en función de mantener una topología deseada.

La Figura 3.4 ilustra dos posibles escenarios que la Inteligencia Colectiva debería administrar. Un primer escenario es un link de comunicación simple presentando una unidad Target y Gateways, conectados mediante dos Gangeway(red de vínculos verde). Un segundo escenario expuesto por la red identificada por sus vínculos de color rojo, cada nodo posee más de un vínculo de comunicación. Una tercera posibilidad es que la red tenga dos unidades objetivos Targets, ejemplo caracterizado por la red de vínculos azules. Es posible identificar en las redes expuestas: Nodos vinculados a dos exploradores GG_{nz} , Nodos que dan servicio de pasarela a dos redes GG_{nx} , Nodos críticos que de perder conectividad inhabilita a la red GG_{np} y Nodos enrutadores pero de carácter estáticos GG_{nh} . La Figura 3.4 expone tres topologías de red puntuales, además se pueden presentar variadas iterconexiones entre los terminales generando nuevos escenarios, esto se debe a que las unidades son móviles.

Dicha inteligencia debe ser capaz de administrar y solucionar temas como: concurrencia, designación de vínculos, Tablas de ruteo, Identificación de unidades redundantes innecesarias y activación de unidades adicionales. Los problemas mencionados, entre otros, son los que se deberán afrontar al diseñar una estrategia de Comportamiento General.

3.2.1. Estructura

Para el programa general se definirán los siguientes puntos:

- Cantidad de Robot Router Autónomo : este parámetro esta dado en el archivo xml utilizado por gazebo denominado **world**, que genera una cantidad fija de Robot Router Autónomo .
- Inteligencia Artificial: consta de una función en el archivo principal que tendrá una entrada(para este Trabajo de Titulo será la potencia de la señal) al sistema de Inteligencia Artificial o Machine Learning programado para cada Robot Router Autónomo .
- Trayectoria del Robot Explorador: Definición del trayecto a realizar por el Robot Explorador.

Comando para Iniciar Simulación:

```
user@user:roslaunch nombre_programa archivo.launcher
```

Ejemplo:

```
user@user:roslaunch Try_1 Try_3.launcher
```

Comando para Dar Ordenes al REA:

```
user@user:rostopic pub /REA std_msgs/String "e REA/archivo_trayectoria"
```

Ejemplo:

```
user@user:rostopic pub /REA std_msgs/String "e REA/zigzag"
```

3.2.2. Diagrama de flujo

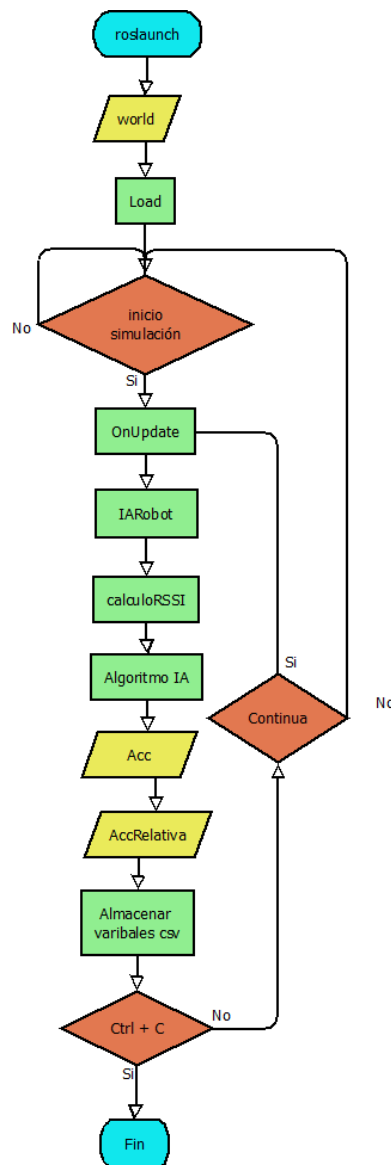


Figura 3.5: Diagrama de Flujo general del programa de simulación.

El esquema de programación propuesto es el mostrado en la Figura 3.5, en el cual se inicia el software de simulación con un comando denominado `roslaunch`, el cual necesita un archivo `*.world`, a través de una función `load` se cargan diferentes modelos de robots, luego se inicia el software gazebo con la simulación detenida, una vez iniciada, avanza a la función `OnUpdate`, que inicia el proceso de IA con la función `IARobot`, a continuación se calcula el RSSI para entregar la entrada al algoritmo de IA (Q-Learning), el cual como



Figura 3.6: TurtleBot.

salida se obtiene una acción que es modificada según la posición relativa del Robot Router Autónomo y, finalmente se almacenan las variables utilizadas en un archivo csv para su posterior análisis en gráficos. Para finalizar el programa se necesita el uso del comando `Ctrl + c` debido al uso de ubuntu como sistema operativo del ordenador. Todas las funciones mencionadas están más detalladas en el Capítulo 5.

3.3. Robots a utilizar

3.3.1. Estación Base(REB)

Debido a que el programa Gazebo no permite dar características funcionales a los objetos, se propone optar por utilizar un robot de forma estática para así representar una estación base de comunicación y poder llevar a cabo la simulación. El robot para cumplir esta función será el mostrado en la Figura 3.6.

3.3.2. Robot Router Autónomo(RRA)

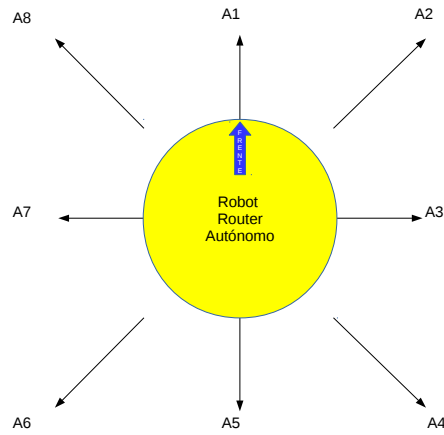


Figura 3.7: Esquema de movimiento del Robot Router Autónomo.



Figura 3.8: iRobot create.

Para efectos prácticos de estudio el Robot Router Autónomo debe ser un robot omni-direccional, sin embargo, no todos los robots omni-direccionales entregados por la comunidad son fiables, es decir, no se comportan de forma omnidireccional pura, por lo que se utilizará un robot con 2 ruedas. Además dado que gazebo puede utilizar robots comerciales reales cuyo URDF(ver Capítulo 4) es compartido por la comunidad, en la mayoría de los casos, se optará por utilizar el robot de la Figura 3.8 que se encuentra en ((2)). Los movimientos del RRA serán los expuestos en la Figura 3.7, incluyendo el movimiento A0 que es detenerse.



Figura 3.9: Pioneer 2DX.

3.3.3. Robot Explorador Autónomo(REA)

Cuando se habla de exploración muchas veces el robot explorador tiene que ser robusto ante escenarios adversos, es por esto que se propone utilizar un robot denominado Ground debido a su existencia comercial ((8)) ver Figura 3.9.

3.4. Código comunicación

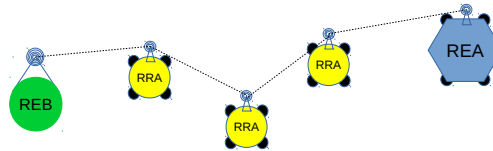


Figura 3.10: Esquema General de Comunicación

La comunicación entre robots será con wifi simulado(ver Capítulo 5, Calcular RSSI). El Robot Estación Base tendrá comunicación directa con solo un robot. Al principio será con el REA y luego sólo con los RRA como muestra la Figura 3.10. Por otro lado el RRA tendrá una comunicación con más de 2 robots al mismo tiempo al ser un Gateway. Finalmente el Robot Explorador tendrá comunicación directa con un Robot pudiendo ser primero el REB y luego solo un RRA para así asegurar una buena comunicación final entre Estación Base y Explorador.

3.4.1. Machine Learning



Figura 3.11: Esquema de funcionamiento de la Inteligencia Artificial.

La Inteligencia Artificial de un robot puede ser modificada sin tener que cambiar el hardware del mismo, es por esto que se propone utilizar un archivo modificable, donde la entrada es la magnitud de la señal RSSI(ver Capítulo 5, Calcular RSSI) y la salida es el movimiento en cualquiera de las direcciones posibles como muestra la Figura 3.11.

Capítulo 4

Lenguajes y aplicaciones utilizadas

*Cualquier estúpido puede escribir código
que una computadora entienda. Los
buenos programadores escriben código
que los humanos entienden.*

Martin Fowler.

RESUMEN: En este capítulo se dará a conocer la estructura general del lenguaje XML y como podemos describir un robot correctamente.

4.1. Lenguaje de programación XML

XML proviene de eXtensible Markup Language (Lenguaje de Marcas Extensible), es un meta-lenguaje ¹ extensible de etiquetas que fue desarrollado por el World Wide Web Consortium (W3C²). Además cabe mencionar que es una adaptación del SGML (Standard Generalized Markup Language), un lenguaje que permite la organización y el etiquetado de documentos. Esto quiere decir que el XML no es un lenguaje en sí mismo, sino un sistema que permite definir lenguajes de acuerdo a las necesidades, como por ejemplo:

- Base de datos
- Documentos de texto
- Hojas de cálculo
- Páginas webs

¹Que se utiliza para decir algo acerca de otro

²<http://www.w3.org>

4.1.1. Ventajas

Algunas de las ventajas que se derivan de la utilización del XML son:

- Ampliable: es posible extender fácilmente el código con la adición de nuevas etiquetas que se ajusten a las necesidades de la aplicación, de modo que se puede continuar utilizando sin ninguna complicación.
- Analizador Estandar: Debido a esto no es necesario crear un analizador específico para cada versión de lenguaje XML. Esto posibilita el empleo de cualquier analizador disponible en la red y así se evitan bugs y se acelera el desarrollo de aplicaciones.
- Estructura Jerárquica: Gracias a esto es sencillo de comprender su estructura y procesarla mejorando la compatibilidad entre aplicaciones.
- Analisis sintactico simple: facil de analizar producto de sus estrictas reglas de composición de un documento.

4.1.2. Estructura de un documento XML

La tecnología XML mantiene la información estructurada de la forma más abstracta y reutilizable posible. Esto quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Por esta razón, presenta una estructura jerárquica. A esas partes se las llaman elementos, y se las señala mediante etiquetas. Una etiqueta es una marca hecha en el documento, que señala un fragmento de éste como un elemento. Las etiquetas tienen la forma `<nombre>`, donde nombre es el nombre del elemento que se está señalando.

4.1.2.1. Prólogo

Es la primera sección del documento XML, son las líneas que indican la versión XML, el tipo de documento y otras cosas. Es opcional y se utiliza en favor de las buenas prácticas. Todo prólogo debe contener:

- Una declaración XML. Es la sentencia que declara al documento como un documento XML.
- Una declaración de tipo de documento. Enlaza el documento con su DTD (definición de tipo de documento), o el DTD puede estar incluido en la propia declaración o ambas cosas al mismo tiempo.
- Uno o más comentarios e instrucciones de procesamiento.

ejemplo: `<?xml version="1.0" encoding="UTF-8">`

4.1.2.2. Cuerpo

Sección obligatoria del documento XML, debe contener un y solo un elemento raíz, característica indispensable también para que el documento esté bien formado. Sin embargo es necesaria la adquisición de datos para su buen funcionamiento **ejemplo:** `<Edit_Mensaje> (...) </ Edit_Mensaje>`

4.1.2.3. Elementos

Un elemento XML es cualquier cosa que esté incluida entre la etiqueta inicial y final. Puede contener texto, atributos, otros elementos o mezcla de los anteriores. Además existe la posibilidad de tener un contenido vacío. Para crear un elemento se debe tener en consideración lo siguiente:

- Son sensibles a letras mayúsculas y minúsculas.
- Deben empezar con una letra o de lo contrario con una barra baja.
- No pueden empezar con las letras "xml".
- Pueden contener letras dígitos, guiones, barras bajas y puntos.
- No pueden contener espacios.

ejemplo:

```
<animal>
  <nombre>Bun</nombre>
  <tipo>León</tipo>
  <color>Marrón</color>
  <edad>15</edad>
  <familia></familia>
</animal>
```

4.1.2.4. Atributos

Los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento. Están diseñados para contener datos relacionados con un elemento específico. Deben ir siempre encerrados entre comillas simples o dobles. Para crear un atributo de manera correcta se debe tener en consideración:

- No pueden contener múltiples valores.
- No pueden contener estructuras en árbol.
- No se pueden expandir fácilmente (para futuros cambios).

ejemplos:

```
<pelicula categoria="accion">
  <titulo idioma="ingles">Mad Max</titulo>
  <director>George Miller</director>
  <estreno>15 mayo 2015</estreno>
  <reparto>Tom Hardy</reparto>
  <reparto>Charlize Theron</reparto>
  <reparto>Nicholas Hoult</reparto>
</pelicula>

<futbolista nombre='Gonzalo "Pipita" Higuaín'>
<futbolista nombre='Gonzalo &quot;Pipita&quot; Higuaín'>
```

4.1.2.5. Entidades predefinidas

Entidades para representar caracteres especiales para que, de esta forma, no sean interpretados como marcado en el procesador XML. ejemplo: Entidad Predefinida: `& amp;`; Caracter `&`

4.1.2.6. Comentarios

Todo documento de desarrollo de aplicaciones necesita contener comentarios del funcionamiento general de algunas sentencias para poder modificar o reemplazar una sección en el futuro por cualquier programador, además del hecho de que es una buena práctica. En particular para documentos XML este comentario, que agrega información no utilizada por el analizador, tiene la siguiente estructura: `<!-- comentario -->`

4.1.3. Sintaxis

Representa las normas a seguir para la construcción de documentos XML dictaminadas por el organismo desarrollador (W3C). Entre ellas destacan:

- El XML es Case - Sensitive: Distingue mayúsculas y minúsculas.
- Todo elemento tiene que tener su correspondiente etiqueta de inicio y de cierre, o una sola etiqueta vacía.
- En Todo documento debe haber un elemento, llamado raíz de documento, que contenga a los demás.
- Todos los elementos deberán estar correctamente anidados.
- Todos los valores de los atributos deberán ir entre comillas.

4.2. XML Robot Description Format(URDF)

Unified Robot Description Format (URDF) es una especificación XML para describir un robot. Tratamos de mantener esta especificación lo más general posible, pero, obviamente, la especificación no puede describir todos los robots. La principal limitación en este punto es que sólo las estructuras de árbol pueden ser representados, descartando todos los robots paralelos. Además, la especificación asume que el robot consiste en eslabones rígidos conectados por articulaciones; elementos flexibles no son compatibles. La especificación incluye:

- Descripción cinemática y dinámica del robot.
- Representación visual.
- Modelo de colisiones.

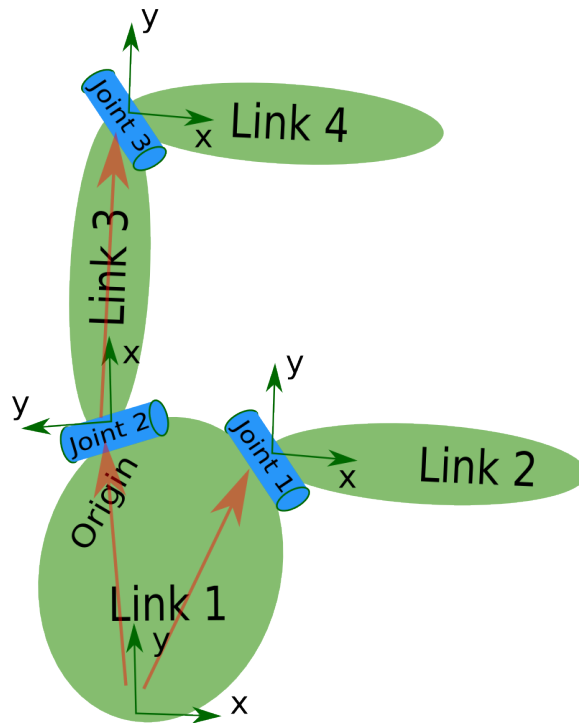


Figura 4.1: Ejemplo de estructura robótica.

La descripción de un robot consiste en un conjunto de elementos de eslabones (links) y de un conjunto de elementos de unión (joints) que conectan los eslabones juntos(ver figura 4.1). ejemplo:

```
<robot name="pr2">
  <link> ... </link>
  <link> ... </link>
  <link> ... </link>
  <joint> .... </joint>
  <joint> .... </joint>
  <joint> .... </joint>
</robot>
```

Se puede ver que el elemento raíz del formato URDF es el elemento `<robot>`. Hay que definir ahora el elemento link y el elemento joint, y las etiquetas y atributos de cada uno de ellos.

4.2.1. Link

Es el elemento descrito por un cuerpo rígido con una inercia y características visuales.

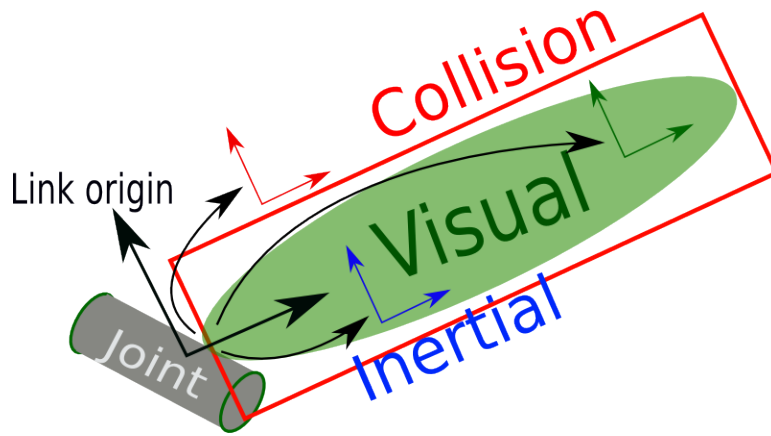


Figura 4.2: Elemento Link

4.2.1.1. Atributos

Name(necesario) El nombre del eslabón(link 4.2).

4.2.1.2. elementos

<inertial> (opcional)

Las propiedades inerciales del eslabón (link).

<origin> (opcional)

Por defecto a la identidad si no se especifica, esta es la posición del marco de referencia inercial, relativo con el marco de referencia del eslabón. El origen del sistema de referencia inercial tiene que estar en el centro de gravedad. Sus ejes no necesitan estar alineados con los ejes principales de la inercia.

xyz (opcional)

Por defecto el vector cero, representa el desplazamiento en x, y y z.

rpy (opcional)

Por defecto a la identidad si no se especifica, representa los ángulos de giro sobre los ejes x, y y z.

<mass> (Kg)

Es el valor de la masa del eslabón.

<inertia> .

La matriz rotacional inercial (3x3), representada en el marco inercial. Debido a que la matriz rotacional inercial es simétrica, solo 6 elementos de esta matriz son especificados, utilizando los atributos *ixx*, *ixy*, *ixz*, *iyx*, *iyz*, *izz*.

<visual> (opcional)

Las propiedades visuales del eslabón (link). Este elemento especifica la forma del objeto (caja, cilindro, esfera, etc.) para propósitos de visualización.

name (opcional)

Especifica un nombre para una parte de la geometría del eslabón. Esto es útil para poder especificar trozos de la geometría.

<origin> (opcional)

Por defecto a la identidad si no se especifica, marco de referencia del elemento visual con respecto al marco de referencia del eslabón.

xyz (opcional)

Por defecto el vector cero, representa el desplazamiento en x, y y z.

rpy (opcional)

Por defecto a la identidad si no se especifica, representa los ángulos de giro sobre los ejes x, y y z.

<geometry> (necesario)

La forma del objeto visual. Puede ser una de las siguientes:

<box> .

El atributo Size representa la longitud de los tres lados del cubo. El origen del cubo está en su centro.

<cylinder> .

Con radius y lenght se especifica su radio y altura. El origen del cilindro está en su centro.

<sphere> .

Especifica el radio con radius. El origen de la esfera está en su centro.

<mesh> .

Una figura tridimensional especificada por un archivo en filename, y un opcional scale que escala el tamaño de la figura. El formato recomendado para la mejor textura y color es el archivo Collada .dae, aunque también soporta archivos .stl.

<material> (opcional)

Es el material del elemento visual. Está permitido especificar un elemento material en el robot (nivel de arriba). Desde el eslabón se puede referir el material por un nombre en name.

<color> (opcional)

rgba El color de un material es definido por un conjunto de cuatro números representando rojo/verde/azul/opacidad, cada uno entre un rango de 0 a 1.

<texture> (opcional)

La textura del material es definida por un archivo, filename.

<collision> (opcional)

Las propiedades colisionales del eslabón (link). Puede ser diferente de las propiedades visuales de un eslabón, por ejemplo, los modelos de colisión más simples son usados para reducir el tiempo de computación.

name (opcional)

Especifica un nombre para una parte de la geometría del eslabón. Esto es útil para poder especificar trozos de la geometría.

<origin> (opcional)

Por defecto a la identidad si no se especifica, véase la descripción **<origin>** en el elemento visual más arriba.

<geometry> .

Véase la descripción **<geometry>** en el elemento visual más arriba.

ejemplo:

```
<link name=\verb|"my_link"|>
  <inertial>
    <origin xyz="0 0 0.5" rpy="0 0 0"/>
    <mass value="1"/>
    <inertia ixx="100" ixy="0" ixz="0" iyy="100" iyz="0" izz="100" />
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <box size="1 1 1" />
    </geometry>
    <material name="Cyan">
      <color rgba="0 1.0 1.0 1.0"/>
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder radius="1" length="0.5"/>
    </geometry>
  </collision>
</link>
```

4.2.2. Joint

Este elemento Joint(articulación) define las propiedades cinemáticas y dinámicas, además de los límites de seguridad de la articulación del robot.

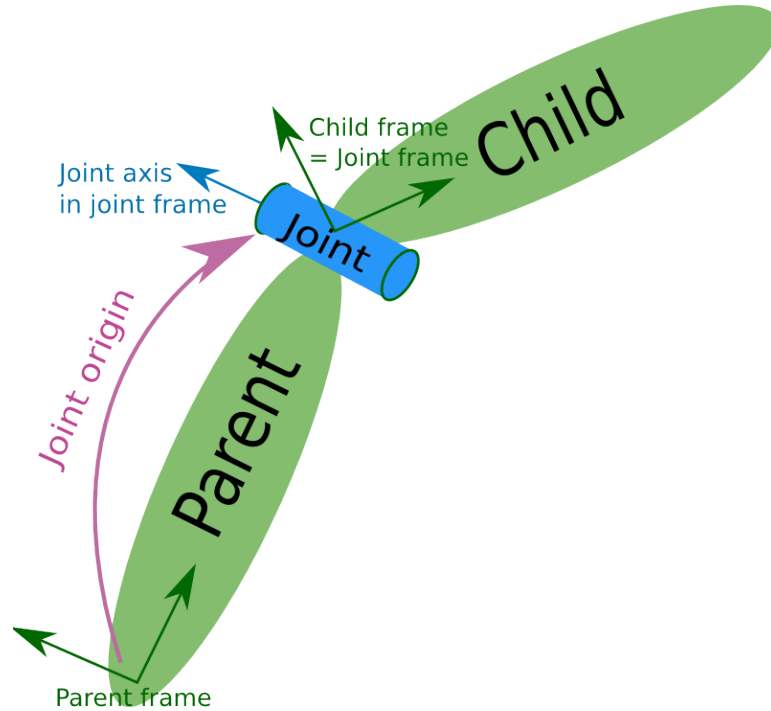


Figura 4.3: Elemento Joint

4.2.2.1. atributos

Name (necesario): Nombre de la articulación.

Type (necesario): Establece el tipo de articulación, los cuales puedes ser uno de los siguientes:

- **revolute**: una articulación de bisagra que gira a lo largo del eje y tiene un rango limitado especificado por los límites superior e inferior.
- **continuous**: una articulación en bisagra continua que gira en torno al eje y no tiene límites superior e inferior.
- **prismatic**: una junta deslizante que se desliza a lo largo del eje, y tiene un rango limitado especificado por los límites superior e inferior.
- **fixed**: Esto no es realmente una articulación, ya que no puede moverse. Todos los grados de libertad están bloqueados. Este tipo

de unión no requiere ejes, la calibración, la dinámica, límites o controlador de seguridad.

- **floating**: Esta articulación permite el movimiento de los 6 grados de libertad.
- **planar**: Esta articulación permite el movimiento en un plano perpendicular al eje.

4.2.2.2. elementos

<origin> (opcional, por defecto la identidad si no se especifica)

Esta es la transformada desde el eslabón del padre con el eslabón del hijo. La articulación se encuentra en el origen del eslabón de hijo.(ver Figura 4.3)

xyz (opcional: por defecto el vector cero)

Representa el desplazamiento en x, y y z.

rpy (opcional: por defecto el vector cero si no se especifica)

Representa la rotación en torno al eje fijo x, y y z. Todos los ángulos se especifican en radianes.

<parent> (necesario)

El nombre del eslabón padre (obligatorio):

link El nombre del eslabón que es el padre de este eslabón en la estructura de árbol del robot.

<child> (necesario)

El nombre del eslabón hijo (obligatorio).

link El nombre del eslabón que es el del hijo.

<axis> (opcional: por defecto (1,0,0))

Este es el eje de rotación de articulaciones de giro, el eje de traslación para juntas prismáticas, y la superficie normal para juntas planas. El eje se especifica en el marco común de referencia. Articulaciones fijas y flotantes no utilizan éste atributo.

xyz (necesario)

Representa el componente de un vector x, y, z. El vector debe estar normalizado.

<calibration> (opcional)

La posición de referencia de la articulación, usada para calibrar la posición absoluta de la articulación.

- **rising** (opcional)
Cuando la articulación se mueve en una dirección positiva, esta posición de referencia dará lugar a un flanco ascendente.
- **falling** (opcional)
Cuando la articulación se mueve en una dirección positiva, esta posición de referencia dará lugar a un flanco descendente.

<dynamics> (opcional)

Un elemento que especifica las propiedades físicas de la articulación. Estos valores se utilizan para definir las propiedades de modelado de la articulación, especialmente útil para la simulación.

- **damping** (opcional, por defecto 0)
El valor de amortiguación física de la articulación ($\frac{N*s}{m}$ para articulaciones prismáticas, $\frac{N*m*s}{rad}$ para articulaciones de revolución).
- **friction** (opcional, por defecto 0)
El valor de fricción estática física de la articulación (N para articulaciones prismáticas, $N*s$ para articulaciones de revolución).

<limit> (únicamente para articulaciones prismáticas y de revoluto)

Un elemento que puede contener los siguientes atributos:

- **lower** (opcional, por defecto es 0)
Un atributo que especifica el límite inferior de unión (radianes para juntas de revolución, metros en las articulaciones prismáticas). En articulaciones continuas no se especifica este campo.
- **upper** (opcional, por defecto es 0)
Un atributo que especifica el límite superior conjunta (radianes para juntas de revolución, metros en las articulaciones prismáticas). En articulaciones continuas no se especifica este campo.
- **effort** (requerido)
Un atributo para el máximo esfuerzo de la articulación ($|\text{esfuerzo aplicado}| < |\text{esfuerzo}|$).
- **velocity** (requerido)
Define la velocidad máxima de la articulación.

<mimic> (opcional)

Esta etiqueta se utiliza para especificar esta articulación imita a otra articulación existent. El valor de este conjunto puede calcularse como $\text{valor} = \text{multiplicador} * \text{otherjointvalue} + \text{offset}$. Va seguido de los siguientes atributos:

joint (requerido)

Especifica el nombre de la articulación a imitar.

multiplier (opcional)

Especifica el factor multiplicativo en la fórmula anterior. El valor por defecto es 1.

offset (opcional)

Especifica el desplazamiento para agregar en la fórmula anterior. Por defecto es 0.

<safety_controller> (opcional)

Un elemento que puede contener los siguientes atributos:

- **soft_lower_limit** (opcional, por defecto es 0)
Un atributo que especifica el límite inferior de la articulación donde el controlador de seguridad comienza a limitar la posición de la articulación. Este límite debe ser mayor que el límite inferior de la articulación.
- **soft_upper_limit** (opcional, por defecto es 0)
Un atributo que especifica el límite superior de la articulación donde el controlador de seguridad comienza a limitar la posición de la articulación. Este límite debe ser menor que el límite superior de la articulación.
- **k_position** (opcional, por defecto es 0)
Un atributo que especifica la relación entre los límites de posición y velocidad.
- **k_velocity** (requerido)
Un atributo que especifica la relación entre el esfuerzo y los límites de velocidad.

ejemplo:

```
<joint name="my_joint" type="floating">
  <origin xyz="0 0 1" rpy="0 0 3.1416"/>
  <parent link="link1"/>
  <child link="link2"/>
  <calibration rising="0.0"/>
  <dynamics damping="0.0" friction="0.0"/>
  <limit effort="30" velocity="1.0" lower="-2.2" upper="0.7" />
  <safety_controller
    k_velocity="10"
    k_position="15"
    soft_lower_limit="-2.0" soft_upper_limit="0.5" />
</joint>
```


4.2.2.3. Archivos Xacro Macro

Existe una forma de generalizar y reutilizar el código para la creación de un robot y para eso existe XACRO. Un XACRO es un tipo de URDF macro que se puede utilizar para generar un URDF. Es muy sencillo después personalizar las partes deseadas y generar el URDF personalizado de todo el robot.

Una de las ventajas de utilizar XACRO es que si hay un error en la especificación XML, en el momento de generar el URDF, el termina indica la línea donde se encuentra dicho error. Si el archivo macro no posee errores, lo generará satisfactoriamente y cuando se ejecute el URDF no mostrará ninguna anomalía.

Ejemplo de XACRO:

Se quiere modelar un robot a partir de su base y su brazo. Para este caso, los archivos que se necesitarán serán:

```
base.xacro union.xacro base_macro.xacro
brazo.xacro brazo_macro.xacro
```

En `base_macro.xacro` y `brazo_macro.xacro` estará todo el código del modelado de esa parte.

En `base.xacro` y `brazo.xacro` se encuentra la referencia al archivo macro de cada parte, es decir:

```
<xacro:include filename="$(find robot)/urdf/base_macro.xacro"/>
<xacro:include filename="$(find robot)/urdf/brazo_macro.xacro"/>
```

En `union.xacro` se encuentra la referencia a estos dos archivos, y la articulación (joint) que une ambas partes. Para generar el URDF de este robot, hay que poner la siguiente instrucción: `roslaunch xacro xacro.py union.xacro robot.urdf`

Donde `robot.urdf` es el programa, que contendrá la especificación XML de las XACROS, pero de una forma más ?limpia y ordenada?. No se recomienda hacer muchos cambios o elaborar directamente el URDF puesto que si después hay algún error en el código será muy difícil detectarlo (el terminal no señala la línea errónea del programa). Es muy importante que la información dentro del URDF sea exacta. Específicamente, los límites de las articulaciones y orientaciones deben ser correctos. Para visualizar el robot hay que ejecutar lo siguiente:

```
roslaunch urdf_tutorial display.launch model:=robot.urdf gui:=True
```

Se abrirá el rviz con el archivo `robot.urdf`.

...

Capítulo 5

Implementación

*Nunca te fíes de un ordenador que no
puedas tirar por la ventana.*

Steve Wozniak.

5.1. Introducción

Cuando se habla de un programa computacional es sabido que contiene archivos y funciones para poder funcionar correctamente, es por esto que a continuación se dará a conocer los distintos archivos involucrados y funciones creadas para llevar a cabo con éxito la simulación del caso estudiado con anterioridad. Los tres archivos de código principales que están guardados en el `Package Try_1` son:

5.1.1. Main

Para el caso de este trabajo se utilizó el nombre `Try_1.cpp` como nombre de archivo principal. Este contiene todas las funciones a utilizar por cada robot para su correcto funcionamiento.

5.1.2. Listener

Archivo que contiene las instrucciones para inicializar el protocolo de comunicación `Publicador/Subscriber` que utiliza ROS y así poder hacer que el robot "`escuche`" instrucciones desde el usuario y "`hable`" su estado para ser observado por el usuario.

5.1.3. Comandos

Archivo encargado de procesar los mensajes realizados por el usuario o robot y así poder paramentrizar, mover y conocer al robot que se desee.

5.2. Estado de los Robots

Para tener un conocimiento de cada robot involucrado en la simulación se crearon funciones que mostrasen el estado de todas las características relevantes del robot en particular, a través de un tópico con la sintaxis `nombreRobot_m` al cual se le debe realizar un `echo`¹ para "**escuchar**" el estado del robot en cuestión.

Ejemplo para REA:

```
rostopic list comando ver topicos iniciados
rostopic echo -c /REA_m comando escuchar características REA
```

Las funciones que se utilizan para el funcionamiento de lo antes expuesto son:

5.2.1. Load

Esta función se encarga de cargar e inicializar los robots del archivo xml `world`, además deja un aviso en la consola del robot iniciado correctamente y su respectivo nombre y también un registro en las variables globales. Esto se genera en el mismo orden del URDF del world.

5.2.2. OnUpdate

Función que es inicializada en Load y llamada cuando se inicia la simulación y utiliza el tiempo de simulación para actualizarse. Además va actualizando el **estado** del robot para poder ser visualizado en una terminal y realiza la llamada a la función `IARobot`. El **estado** incluye los siguientes datos que pueden ser observados en pantalla:

- Id del robot.
- Nombre del robot
- Ángulo de las uniones.
- Velocidad de las uniones.
- Posición en x del robot
- Posición en y del robot
- Modulo de la posicion con respecto al Robot Estación Base
- Rssi relativo al Robot Estación Base
- Accion realizada por el robot

¹comando para la impresión de un texto en pantalla. Es utilizado en las terminales de los sistemas operativos como Unix, GNU/Linux, o MS-DOS; dentro de pequeños programas llamados scripts; y en ciertos lenguajes de programación tales como PHP.

5.2.3. **buscar**

Función encargada de encontrar todas las uniones del robots.

5.2.4. **pintar**

Función que muestra por terminal el listado de las uniones del robot.

5.2.5. **cargarUniones**

Función encargada de establecer las uniones del robot con valores por defecto.

5.3. **Cantidad de Robots Router Autónomos**

El programa creado se inicia desde un archivo *.launcher que está asociado a un *.world el cual es un archivo XML con etiquetas bien definidas por la comunidad de URDF, es decir, que un archivo antes modificado es leído para realizar la simulación. Es por esto que, en este apartado, se dará a conocer la manera de adicionar Robot Router Autónomo, las variables globales a utilizar, el código de Inteligencia Artificial que utiliza cada robot intrínsecamente y la manera de auto-gestionar un mensaje enviado desde la estructura de ROS, la cual es entre subscriptor y publicador.

5.3.1. Adicionar RRA

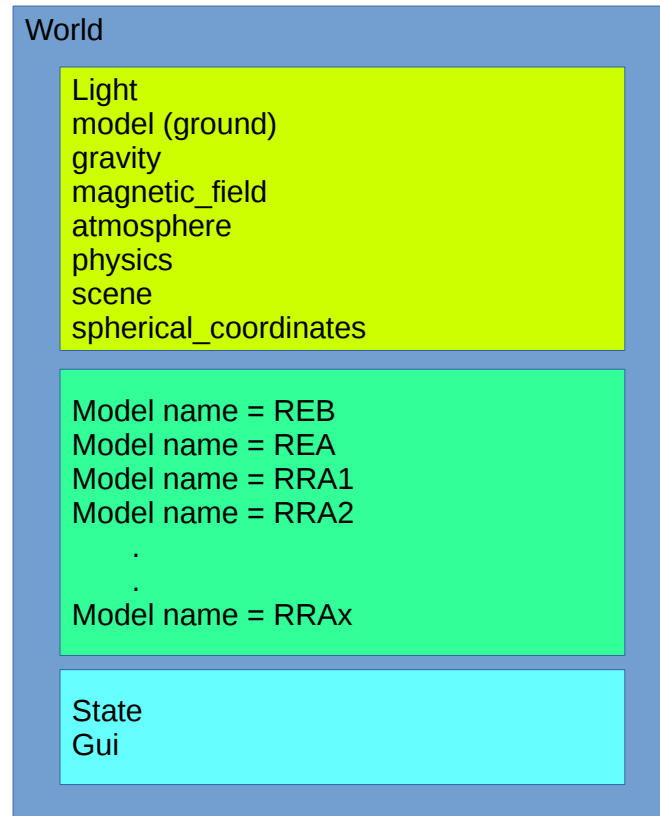


Figura 5.1: Estructura de un archivo *.world.

Para poder adicionar un Robot Router Autónomo se debe conocer la estructura general de un *.world la cual esta representada por la Figura 5.1. En esta estructura el código URDF de los robots va uno a continuación del otro, y para tener un orden en el programa se deberá incluir primero el Robot Estación Base , seguido de Robot Explorador Autónomo y finalmente los Robot Router Autónomo, dando un número ascendente al final del nombre RRA, comenzando desde el 1. Luego se debe incluir el sistema operativo o plugins a utilizar, que para efectos del código entregado en este trabajo, lleva por nombre `name = "actuador"` y el `filename = "libTry_1.so"`, que se encuentra en el directorio `devel/lib` y se coloca al final de cada URDF agregado como se muestra en la Figura 5.2.

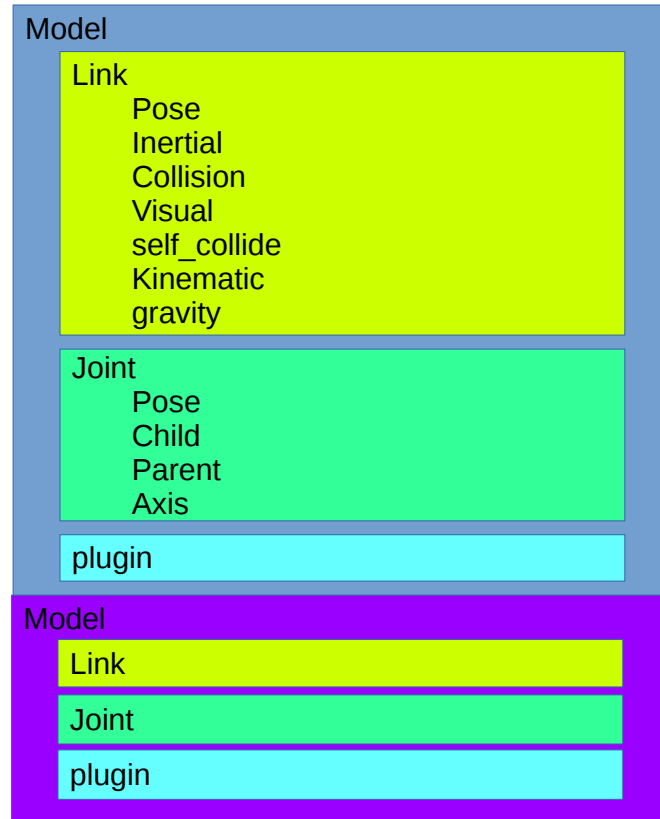


Figura 5.2: Estructura de model de un robot.

5.3.2. Variables Globales Auxiliares

Debido a que el sistema operativo creado para los robots es el mismo en cada uno de ellos (`libTry_1.so`), se deben utilizar variables auxiliares debidamente indexadas y que se puedan guardar en cada momento para su uso en las funciones creadas.

- `int conexiones` : constante utilizada para llevar el conteo de cuantos robots se inicializan.
- `vector<int> id_robot` : vector que contiene el id del modelo del robot iniciado.
- `vector<std::string> accion_robot` : Variable auxiliar para evitar un movimiento de retroceso consecutivo en la desición de acción mediante

la Inteligencia Artificial implementada.

- `vector<std::string> robots_names` : almacena en orden el nombre de los robots iniciados REB->REA->RRA1->...->RRAx.
- `vector<double> modulo_robots` : vector que almacena el módulo de la posición (x, y) de cada robot de la cadena y es calculado con la siguiente fórmula:

$$modulo_{robot[i]} = \sqrt{x^2 + y^2}$$

- `vector<std::string> tipo_robot` : vector que contiene el tipo de robot a analizar para seleccionar la carpeta de instrucciones correspondientes a dicho robot:
 - REB
 - REA
 - RRA

- `vector<double> vec_x` : vector que almacena el valor de la posición en x de cada robot.
- `vector<double> vec_y` : vector que contiene el valor de la posición en y de cada robot.
- `vector<double> vec_dif` : vector que almacena la diferencia de RSSI entre el robot superior(sucesor) y robot inferior (antecesor) según la fórmula:

$$vec_{dif_i} = RSSI_{sup} - RSSI_{inf}$$

- `vector<double> vec_sup` : vector que contiene la RSSi actual del robot superior(sucesor) de cada robot.
- `vector<double> vec_inf` : vector que contiene la RSSi actual del robot inferior(antecesor) de cada robot.
- `vector<int> vec_Minvc` : vector que almacena el valor del indicador Min_{vc} de cada robot.
- `vector<int> vec_Maxvl` : vector que almacena el valor del indicador Max_{vl} de cada robot.
- `vector<int> vec_MinDif` : vector que almacena el valor del indicador Min_{Difa} de cada robot.
- `vector<int> vec_SS` : vector que almacena el valor del indicador $SS_{(k)}$ de cada robot.

- **vector<int> vec_Acc** : vector que almacena el valor del indicador *Acc* de cada robot.
- **vector<int> vec_Accr** : vector que almacena el valor del indicador *Acc_r* de cada robot.
- **vector<double> R** : vector que almacena el valor de la Recompensa de cada robot.
- **double Robot_sucesor** : variable que almacena el valor del módulo del robot sucesor de cada uno de los robots y se reutiliza al ser global. Es calculado con la siguiente fórmula:

$$Robot_{sucesor} = modulo_{robots[i-1]} - modulo_{robots[i]}$$

- **double Robot_antecesor** : variable que almacena el valor del módulo del robot antecesor de cada uno de los robots y se reutiliza al ser global. Es calculado con la siguiente fórmula:

$$Robot_{antecesor} = modulo_{robots[i]} - modulo_{robots[i+1]}$$

- **double Q [10] [2] [10]** : matriz para almacenar los valores de *Q* de cada robot.
- **int AccRelativa [8] [8]** : matriz que contiene los valores para tomar una acción relativa entre Orientación del robot y el valor de la Acción correspondiente a la matriz *Q*.
- **vector<int> contador** : variable utilizada para realizar el envío de una instrucción y va aumentando cada vez que se cumple un ciclo de simulación del programa.

5.3.3. IARobot

Función encargada de:

- Actualizar los vectores globales con los datos correspondientes de cada robot.
- Utilizar las funciones de calcularRSSI, Accion y enviarRos cuando se necesite según la lógica planteada a través de funciones **if** y **for**.

5.3.4. enviarRos

Con esta función se puede publicar una orden el mismo robot con el formato de comunicación de ROS, recibiendo un **publicador** y un **mensaje**.

5.4. Inteligencia Artificial

La inteligencia de los robots simulados depende del tipo de robot pero, para simplificar el código y los archivos involucrados, se utiliza un mismo código que reacciona distinto en cada tipo de robot. Sin embargo, existen funciones comunes para poder llevar a cabo una decisión que, en este, caso es la de calcular RSSI.

5.4.1. calcularRSSI

Toda Inteligencia Artificial necesita un parámetro de entrada, como estimulación, para tomar una decisión, en este caso es el valor RSSI. Para simplificar la creación del programa de simulación el cálculo de RSSI se hace utilizando el módulo de la ubicación de los robots y luego se resta una pérdida asociada a la distancia entre el receptor y emisor de la señal de comunicación. Esto se ve reflejado en la ecuación:

$$RSSI_{relativo} = Potencia_{transmisin} - Perdida_{Gaussiana}$$

y la pérdida Gaussiana se calcula con la siguiente ecuación ²:

$$10 * \gamma * \log_{10}(mdulo_{robot} / l_{puntoreferencia}) + e^{-1 * rand()^2} (RuidoGaussiano)$$

5.4.2. Accion

Esta función se encarga de escoger el archivo Ax que se debe realizar el Robot Router Autónomo según la Inteligencia Artificial seleccionada y que para este caso es Q-Learning. Para esto se siguieron los estudios de Palma que tiene las siguientes variables:

- Dif_n :Diferencia de RSSI entre el robot sucesor y el robot antecesor.
- Dif_a :Módulo de Dif_n calculado de la ecuación: $\sqrt{(RSSI_{sup}^2 + RSSI_{inf}^2)}$
- Dif_d :Diferencia digital RSSI donde +1 significa más cercano al robot superior y -1 más cercano al robot inferior
- Max_{vl} :Maximizar vínculo lejano.

$$MAX_{vl} = 1; SI(RSSI_{lejano}(k) - RSSI_{lejano}(k-1) < 0)$$

$$MAX_{vl} = 0; SI(RSSI_{lejano}(k) - RSSI_{lejano}(k-1) > 0)$$

- Min_{vc} :Minimizar vínculo cercano.

$$MIN_{vc} = 1; SI(RSSI_{cercano}(k) - RSSI_{cercano}(k-1) < 0)$$

$$MIN_{vc} = 0; SI(RSSI_{cercano}(k) - RSSI_{cercano}(k-1) > 0)$$

²<https://www.gaussianwaves.com/2013/09/log-distance-path-loss-or-log-normal-shadowing-model/>

Algoritmo de obtención de Maxvl y Minvc:

```

if (RSSI_sup > RSSI_inf)
    if (vec_sup[i] - vec_sup_a[i] < 0)
        Minvc = 1
    else
        Minvc = 0

    if (vec_inf[i] - vec_inf_a[i] > 0)
        Maxvl=1;
    else
        Maxvl=0
else
    if (vec_inf[i] - vec_inf_a[i] < 0)
        Minvc = 1
    else
        Minvc = 0

    if (vec_sup[i] - vec_sup_a[i] > 0)
        Maxvl=1
    else
        Maxvl=0

```

- Min_{Dif} Minimizar diferencia de RSSI entre el robot sucesor y el robot antecesor.

$$MIN_{Dif} = 1; SI(Dif_a(k) - Dif_a(k-1) < 0)$$

$$MIN_{Dif} = 0; SI(Dif_a(k) - Dif_a(k-1) > 0)$$

- $R_{(Min_{Dif}, Max_{vl}, Min_{vc})}$: Recompensa con valor 1 para estado SM1 y con valor -1 para los estados SM2 al SM8. según la tabla 5.1

Tabla 5.1: Cumplimiento de objetivos

Estado	MIN_{Dif}	MIN_{vc}	MAX_{vl}
SM1	1	1	1
SM2	1	1	0
SM3	1	0	1
SM4	1	0	0
SM5	0	1	1
SM6	0	1	0
SM7	0	0	1
SM8	0	0	0

5.4.3. getDeseado

Tabla 5.2: Accion deseada v/s Orientacion robot

$Orientacin_{Robot}$	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
A_1	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
A_2	A_8	A_1	A_2	A_3	A_4	A_5	A_6	A_7
A_3	A_7	A_8	A_1	A_2	A_3	A_4	A_5	A_6
A_4	A_6	A_7	A_8	A_1	A_2	A_3	A_4	A_5
A_5	A_5	A_6	A_7	A_8	A_1	A_2	A_3	A_4
A_6	A_4	A_5	A_6	A_7	A_8	A_1	A_2	A_3
A_7	A_3	A_4	A_5	A_6	A_7	A_8	A_1	A_2
A_8	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_1

Debido a que en el algoritmo original analizado se utilizó un Robot Omnidireccional para los Robot Router Autónomo y en la simulación creada en este trabajo se utiliza un Robot No Omnidireccional, se debe utilizar esta función que identifica la orientación del robot y realiza un movimiento acorde a la selección de la Inteligencia Artificial, es decir, se tiene una acción deseada y se busca obtener la acción del robot según su orientación que sea equivalente a la acción deseada según la Tabla 5.2

5.5. Trayectoria del Robot Explorador Autónomo

Para poder manejar las distintas partes de los robots involucrados en la simulación se definieron algunas funciones que ayudan a controlar cada parámetro, así como velocidad, ángulo, PID³ entre otros. Además se estandarizó la carpeta de las instrucciones a utilizar y los movimientos de los tipos de robots utilizados. Las funciones que nos ayudan a realizar esta tarea se describirán a continuación.

5.5.1. procesar

Esta función que se encuentra en el archivo `Comandos.cpp` está encargada de recibir el mensaje enviado por el usuario o desde el mismo robot a través del protocolo de comunicación de ROS (mediante la publicación a un tópico), utilizando la función `Switch/Case` obteniendo como parámetros:

- d : se utiliza para mostrar en el "estado" del robot las partes del robot.
- p : se utiliza para cambiar los parámetros de velocidad de las partes del robot.
- m : se utiliza para mover una parte del robot a un ángulo específico.

³ valores de controlador proporcional, integral y derivativo de un motor de corriente continua

- `s` : sirve para provocar un tiempo de espera de las instrucciones del robot, medido en milisegundos y así conseguir crear una trayectoria al robot.
- `e` : se utiliza para abrir un archivo con instrucciones definidas por el usuario y tiene por defecto la carpeta `/Robot/src/Try_1/instrucciones/`, por lo que sólo necesita el tipo de robot y el nombre del archivo como se ejemplifica a continuación:

En general:
`tipoRobot/archivo`

Ejemplo
`RRA/A0`

5.5.2. mover

Esta función fue creada para mover un Joint a un ángulo específico medido en radianes utilizando la siguiente ecuación:

$$Error = |Angulo_{actual} - Angulo_{deseado}|$$

con el Error calculado se actualiza el PID del Joint el cual tiene por defecto las siguientes constantes:

- Acción proporcional : `p = 2`
- Acción Integral : `i = 1`
- Acción derivativa : `d = 1`
- Limite superior Integral : `imax = 0`
- Limite inferior Integral : `imin = 0`
- Valor maximo de salida : `cmdMax = Velocidad del URDF`
- Valor minimo de salida : `cmdMin = -1 * Velocidad del URDF`

5.5.3. parametrizar

Es la función encargada de establecer la Velocidad de una union del robot medida en `[metros/segundos]` y cambiar la velocidad máxima del PID del robot. La sintaxis es:

Para setear velocidad en general:
`p nombre_exacto_union_robot V velocidad_deseada tiempoAccion`
 Ejemplo:

```
p create::left_wheel V 0 1
```

Para modificar la Velocidad Maxima En general:

```
p nombre_exacto_union_robot VM velocidadMaxima_deseada 0
```

Ejemplo:

```
p create::left_wheel VM 2 0
```

5.5.4. Trayectoria zigzag

Gracias a las funciones creadas y mencionadas anteriormente, se utilizó un archivo llamado **zigzag** con las instrucciones de la Tabla 5.3 para que el Robot Explorador Autónomo tenga una trayectoria en forma de zigzag y así poder analizar la Inteligencia Artificial utilizada por los Robots Routers Autónomos .

Tabla 5.3: Contenido Archivo zigzag

Parámetro	Instrucción	Explicación
e	REA/A3	movimiento A3
s	9000	espera 9 segundos
e	REA/A1	movimiento A1
s	40000	espera 40 segundos
e	REA/A7	movimiento A7
s	18000	espera 18 segundos
e	REA/A1	movimiento A1
s	40000	espera 40 segundos
e	REA/A3	movimiento A3
s	18000	espera 18 segundos
e	REA/A1	movimiento A1
s	40000	espera 40 segundos
e	REA/A7	movimiento A7
s	18000	espera 18 segundos
e	REA/A1	movimiento A1
s	40000	espera 40 segundos
e	REA/A0	movimiento A0

5.5.5. Trayectoria L Derecha

Gracias a las funciones creadas y mencionadas anteriormente, se utilizó un archivo llamado **Lder** con las instrucciones de la Tabla 5.4 para que el Robot Explorador Autónomo tenga una trayectoria en forma de diagonal derecha y así poder analizar la Inteligencia Artificial utilizada por los Robots Routers Autónomos .

Tabla 5.4: Contenido Archivo diader

Parámetro	Instrucción	Explicación
e	REA/A1	avanzar
s	120000	espera 120 segundos
e	REA/A3	girar 90°
s	51200	espera 51.2 segundos
e	REA/A1	avanzar
s	120000	espera 120 segundos
e	REA/A0	detener

5.6. Iniciar Simulación

Para poder comenzar a utilizar el programa creado se debe configurar las referencias de ROS desde la carpeta correspondiente al código a utilizar. En este caso y para ejemplificar se utilizó la carpeta denominada "Robot", la cual contiene las carpetas creadas por `carkin_make`, "build", "src" y "devel". La carpeta devel contiene un archivo de configuración llamado "setup.sh" el cual se debe iniciar de la siguiente forma:

```
cd Robot/
source devel/setup.sh
```

Esto da paso a poder utilizar todas las funciones de ROS desde la consola para su funcionamiento y se debe realizar en cada una de las ventanas de consolas que se quiera utilizar. Luego se debe iniciar la simulación con el comando `ros roslaunch` seguido del package (`Try_1`) creado que contiene el código fuente a utilizar y finalmente el archivo *.launcher que contiene el *.world a utilizar.

```
roslaunch Try_1 Try_3.launcher
```

Capítulo 6

Análisis de Resultados

*Los científicos de hoy piensan
profundamente en lugar de claramente.
Se debe estar cuerdo para pensar con
claridad, pero se puede pensar
profundamente y estar completamente
loco.*

Nikola Tesla

RESUMEN: En el capítulo a presentar se analizarán los resultados de la simulación hecha con gazebo, ROS y Cpp en relación al uso del recurso computacional, la visualización del programa y análisis de variables con gráficos.

6.1. Introducción

Como es de esperar para un programa de simulación se debe analizar los distintos aspectos que lo involucra la ejecución de este. Es por esto que se mostrará a continuación el resultado del análisis de el recurso computacional utilizado, la visualización final de la simulación y las variables obtenidas.

6.2. Recurso computacional

Como Hardware principal se utilizó una tablet con sistema operativo Linux con las siguientes especificaciones de software y hardware:

- Distribución: Ubuntu 16.04 LTS.
- Memoria: 4 GB.

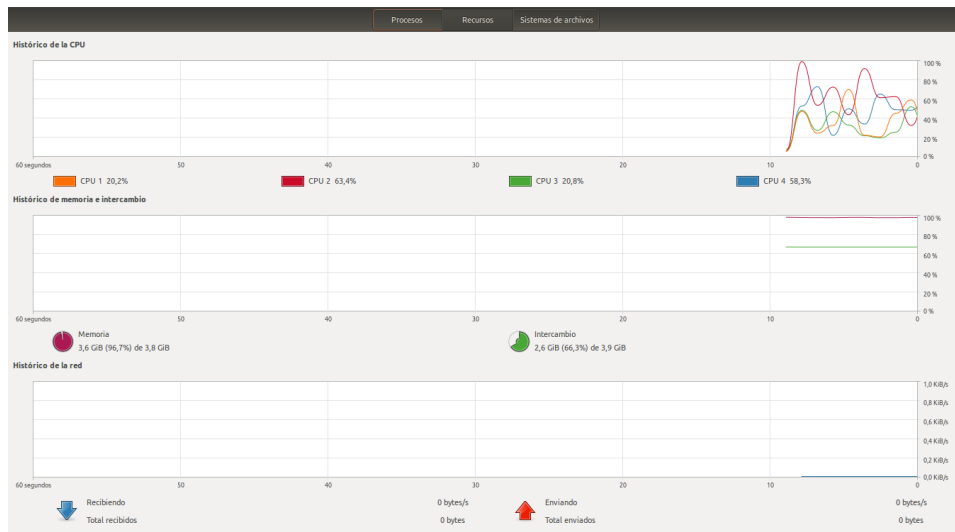


Figura 6.1: Información de Monitor de Sistema con programa detenido.

- Procesador: Intel Core M5Y10c CPU 800MHz x 4
- Gráficos: Intel HD Graphics 5300 (Broadwell GT2).
- Tipo de SO: 64 bits.
- Disco: SSD 64 GB.

6.2.1. Análisis Monitor de Sistema

Para poder analizar el rendimiento del programa creado se debe utilizar una herramienta que revise el estado y funcionamiento del sistema utilizado, es por esto que se utilizó el software **Monitor de Sistemas** que trae por defecto la distribución Linux Ubuntu. Se analizaron cinco estados del programa detenido, iniciado, con un Robot Router Autónomo funcionando, con dos Robot Router Autónomo funcionando y con tres Robot Router Autónomo funcionando. Los resultados fueron los siguientes:

- Simulación detenida: Al iniciar el programa con 3 Robot Router Autónomo y un Robot Explorador Autónomo el equipo tiene un consumo disparado de los procesadores entre 20 y 60 por ciento de la capacidad de los procesadores y un 95.4 y 66 por ciento de la memoria Ram e intercambio, respectivamente, como se observa en el gráfico de la Figura 6.1.
- Simulación iniciada: Una vez iniciada la simulación se puede ver el aumento del uso de los procesadores a un rango entre un 72 y 94

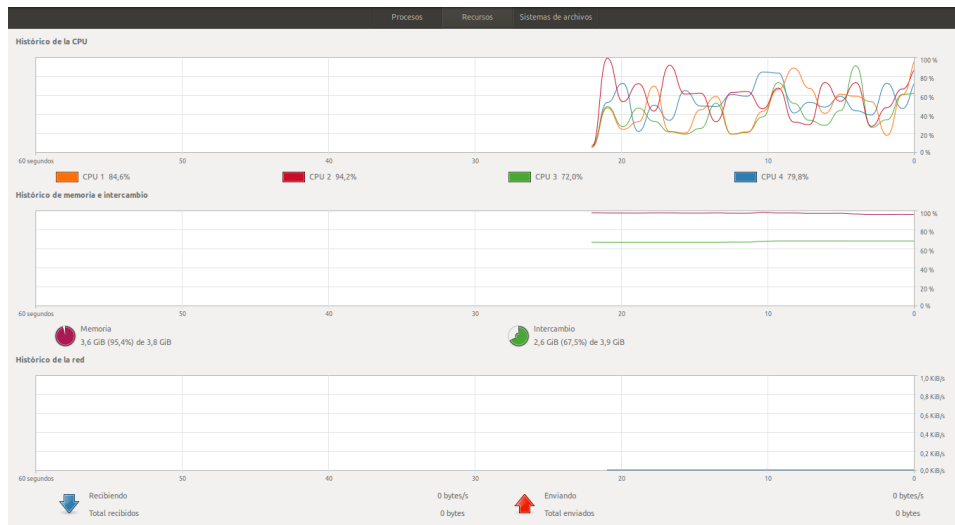


Figura 6.2: Información de Monitor de Sistema con programa iniciado.

por ciento y un aumento de memoria Ram mínimo como se visualiza en la Figura 6.2.

- Un Robot Router Autónomo desplegado: Una vez alcanzada las condiciones necesarias para poner en funcionamiento al primer Robot Router Autónomo se observa el uso completo de un procesador, otro al 92 y los dos restantes a 70 por ciento, las memorias de Ram e intercambio no varían mucho como se muestra en la Figura 6.3.



Figura 6.3: Información de Monitor de Sistema con programa con 1 Robot Router Autónomo .

- Dos Robots Routers Autónomos desplegados: Una vez alcanzada las condiciones necesarias para poner en funcionamiento al segundo Robot Router Autónomo se observa el uso de procesador en 83,71,92 y 83 por ciento, las memorias de Ram disminuye y el intercambio aumenta como se muestra en la Figura 6.4.
- Tres Robots Routers Autónomos desplegados: Una vez alcanzada las condiciones necesarias para poner en funcionamiento al tercero Robot Router Autónomo se observó el uso de procesador cercano al 90 por ciento, las memorias de Ram e intercambio se mantiene en realción al anterior.



Figura 6.4: Información de Monitor de Sistema con programa con 2 Robot Router Autónomo .

6.2.2. Información por consola

Debido a que el programa funciona en terminales de consola del sistema, se aprovechó en utilizar este medio para ir observando la variación Online de los indicadores de cada uno de los robots, de manera individual.

```

[INFO] [155699790.185289122]: Finished loading Gazebo ROS API Plugin.
[Msg] Waiting for master.
[INFO] [155699790.186822449]: waitForService: Service [/gazebo/set_physics_properties] has not been advertised, waiting...
[Msg] Published address: 192.168.0.15
[Msg] [msgs.cc:1655] Conversion of sensor type[depth] not supported.
[Msg] [Try_1.cpp:56] Iniciacion Correcta de R2
[Msg] [Try_1.cpp:56] Iniciacion Correcta de R2A
[Msg] [Try_1.cpp:56] Iniciacion Correcta de R2A1
[Msg] [Try_1.cpp:56] Iniciacion Correcta de R2A2
[Msg] [Try_1.cpp:56] Iniciacion Correcta de R2A3
[Msg] [msgs.cc:1655] Conversion of sensor type[depth] not supported.
[Msg] [msgs.cc:1655] Conversion of sensor type[depth] not supported.
[Msg] [msgs.cc:1655] Conversion of sensor type[depth] not supported.
[INFO] [155699854.472325516, 0.061880000]: waitForService: Service [/gazebo/set_physics_properties] is now available.
[INFO] [155699854.472325516, 0.061880000]: Physics dynamic reconfigure ready.
[Msg] [listener.cpp:46] e RRA1/As
[Msg] [Try_1.cpp:368] RRA1:
Mindf es: 1, Maxvl es: 1, Minvc es: 1
R2S1_sup es: 15.233028, R2S1_inf es: 20.339085, Difa es: 5.103977
Direccion es: 0, Acc es: 1[Msg] [Try_1.cpp:377] ,Acc_relative es: 1
[Msg] [Try_1.cpp:368] RRA1:
Mindf es: 1, Maxvl es: 1, Minvc es: 1
R2S1_sup es: 15.278367, R2S1_inf es: 20.154580, Difa es: 4.876213
Direccion es: 0, Acc es: 1[Msg] [Try_1.cpp:377] ,Acc_relative es: 1
[Msg] [Try_1.cpp:368] RRA1:
Mindf es: 1, Maxvl es: 1, Minvc es: 1
R2S1_sup es: 15.322081, R2S1_inf es: 19.974208, Difa es: 4.654127
Direccion es: 0, Acc es: 1[Msg] [Try_1.cpp:377] ,Acc_relative es: 1
[Msg] [listener.cpp:46] e RRA1/As
[Msg] [Try_1.cpp:368] RRA1:
Mindf es: 1, Maxvl es: 1, Minvc es: 1
R2S1_sup es: 15.338576, R2S1_inf es: 19.886126, Difa es: 4.555551
Direccion es: 0, Acc es: 1[Msg] [Try_1.cpp:377] ,Acc_relative es: 1
[Msg] [listener.cpp:46] e RRA1/As
[Msg] [Try_1.cpp:368] RRA1:
Mindf es: 1, Maxvl es: 1, Minvc es: 1
R2S1_sup es: 15.337732, R2S1_inf es: 19.800628, Difa es: 4.462896
Direccion es: 0, Acc es: 1[Msg] [Try_1.cpp:377] ,Acc_relative es: 1
[Msg] [listener.cpp:46] e RRA1/As
[Msg] [Try_1.cpp:368] RRA1:
Mindf es: 1, Maxvl es: 1, Minvc es: 1
R2S1_sup es: 15.344872, R2S1_inf es: 19.716487, Difa es: 4.371615
Direccion es: 0, Acc es: 1[Msg] [Try_1.cpp:377] ,Acc_relative es: 1
[Msg] [Try_1.cpp:368] RRA1:
Mindf es: 1, Maxvl es: 1, Minvc es: 1
R2S1_sup es: 15.352022, R2S1_inf es: 19.633649, Difa es: 4.281627
Direccion es: 0, Acc es: 1[Msg] [Try_1.cpp:377] ,Acc_relative es: 1
[Msg] [Try_1.cpp:368] RRA1:
Mindf es: 1, Maxvl es: 1, Minvc es: 1
R2S1_sup es: 15.359160, R2S1_inf es: 19.552856, Difa es: 4.192890
Direccion es: 0, Acc es: 1[Msg] [Try_1.cpp:377] ,Acc_relative es: 1
[Msg] [Try_1.cpp:368] RRA1:
Mindf es: 1, Maxvl es: 1, Minvc es: 1
R2S1_sup es: 15.366292, R2S1_inf es: 19.471722, Difa es: 4.105430
Direccion es: 0, Acc es: 1[Msg] [Try_1.cpp:377] ,Acc_relative es: 1
[Msg] [Try_1.cpp:368] RRA1:
Mindf es: 1, Maxvl es: 1, Minvc es: 1
R2S1_sup es: 15.373435, R2S1_inf es: 19.392576, Difa es: 4.019161
Direccion es: 0, Acc es: 1[Msg] [Try_1.cpp:377] ,Acc_relative es: 1
[Msg] [listener.cpp:46] e RRA1/As
[Msg] [Try_1.cpp:368] RRA1:
Mindf es: 1, Maxvl es: 1, Minvc es: 1
R2S1_sup es: 15.380554, R2S1_inf es: 19.314582, Difa es: 3.934048
Direccion es: 0, Acc es: 1[Msg] [Try_1.cpp:377] ,Acc_relative es: 1
[Msg] [Try_1.cpp:368] RRA1:
Mindf es: 1, Maxvl es: 1, Minvc es: 1
R2S1_sup es: 15.387656, R2S1_inf es: 19.237715, Difa es: 3.850059
Direccion es: 0, Acc es: 1[Msg] [Try_1.cpp:377] ,Acc_relative es: 1
[Msg] [listener.cpp:46] e RRA1/As
[Msg] [Try_1.cpp:368] RRA1:
Mindf es: 1, Maxvl es: 1, Minvc es: 1
R2S1_sup es: 15.394783, R2S1_inf es: 19.161833, Difa es: 3.767351
Direccion es: 0, Acc es: 1[Msg] [Try_1.cpp:377] ,Acc_relative es: 1
[Msg] [listener.cpp:46] e RRA1/As
[Msg] [Try_1.cpp:368] RRA1:

```

Figura 6.5: Información mostrada a través de debug por consola.

6.2.2.1. Debug Gazebo

Una forma efectiva para destacar un mensaje por terminal, se utiliza el comando `gzdbg`, que muestra el mensaje en color celeste, para publicar las variables del robot RRA1, como ejemplificación, en el Figura 6.5.

```

mrmelkix@mrmelkix: ~/Robot
data: Hay conectados
Elementos

El numero del modelo principal es:727
el nombre del model es: RRA1
left_wheel: Ang: 0.001765 Vel: -0.000006
, x: 0.448164, y: 0.009584, yaw[radian]: -0.000006: mod 0.448267
Rssi Relativa: 18.021563[dB] Accion: Aun no se necesita un robot Matriz Q de RRA1 es:
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
Matriz Q de RRA2 es:
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
Matriz Q de RRA3 es:
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

8.233000

El numero del modelo principal es:727
el nombre del model es: RRA1
right_wheel: Ang: 0.001740 Vel: -0.000006
, x: 0.448164, y: 0.009584, yaw[radian]: -0.000006: mod 0.448267
Rssi Relativa: 18.021563[dB] Accion: Aun no se necesita un robot Matriz Q de RRA1 es:
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
Matriz Q de RRA2 es:
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
Matriz Q de RRA3 es:
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

8.233000

```

Figura 6.6: Información mostrada por consola a través de un topic.

6.2.2.2. Estado robots

Otra opción para visualizar es escuchando los mensajes de tópicos, que para ejemplificar, se utilizó la escucha del tópico `RRA1_m` como muestra la Figura 6.6.

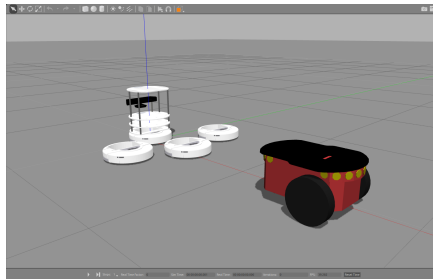
6.3. Visualización

Para poder realizar el análisis visual se divide en funcionamiento del programa en 6 imágenes desde el inicio hasta el final tomando screenshots del programa.

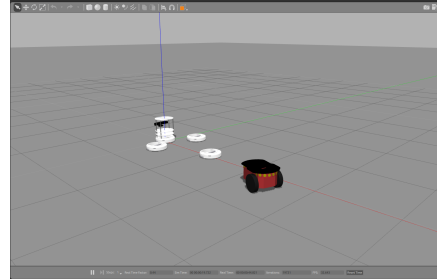
6.3.1. Análisis trayectorias

Para demostrar que el programa funciona correctamente se simularon dos trayectorias con tres Robot Router Autónomo y un Robot Explorador Autónomo las cuales se observarán a continuación.

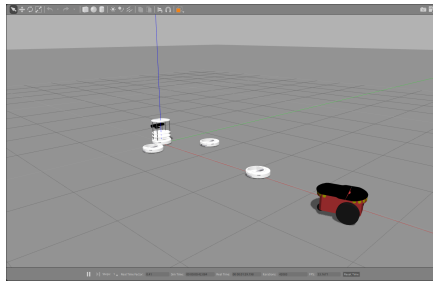
6.3.1.1. Trayectoria zigzag



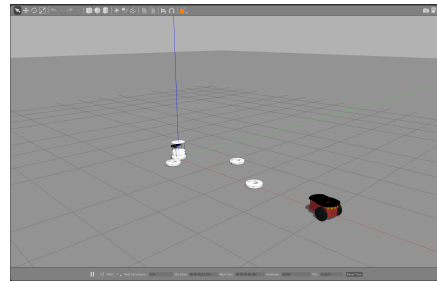
(a) posición inicial



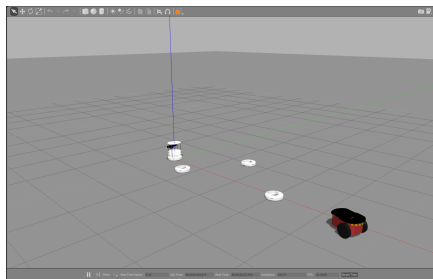
(b)



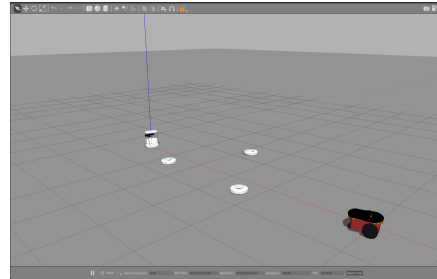
(c)



(d)



(e)



(f) posición final

Figura 6.7: Ejecución trayectoria zigzag

Para el primer experimento se utilizó la trayectoria en zigzag mostrada en la Figura 6.7 . La parte (a) muestra la distribución inicial del experimento, (b) muestra el despliegue del primer Robot Router Autónomo , (c) muestra el despliegue del segundo Robot Router Autónomo , (d) muestra el despliegue del tercer Robot Router Autónomo , (e) muestra la distribución casi al final del experimento y (f) muestra la posición de cada Robots al terminar el experimento.

6.3.1.2. Trayectoria L derecha

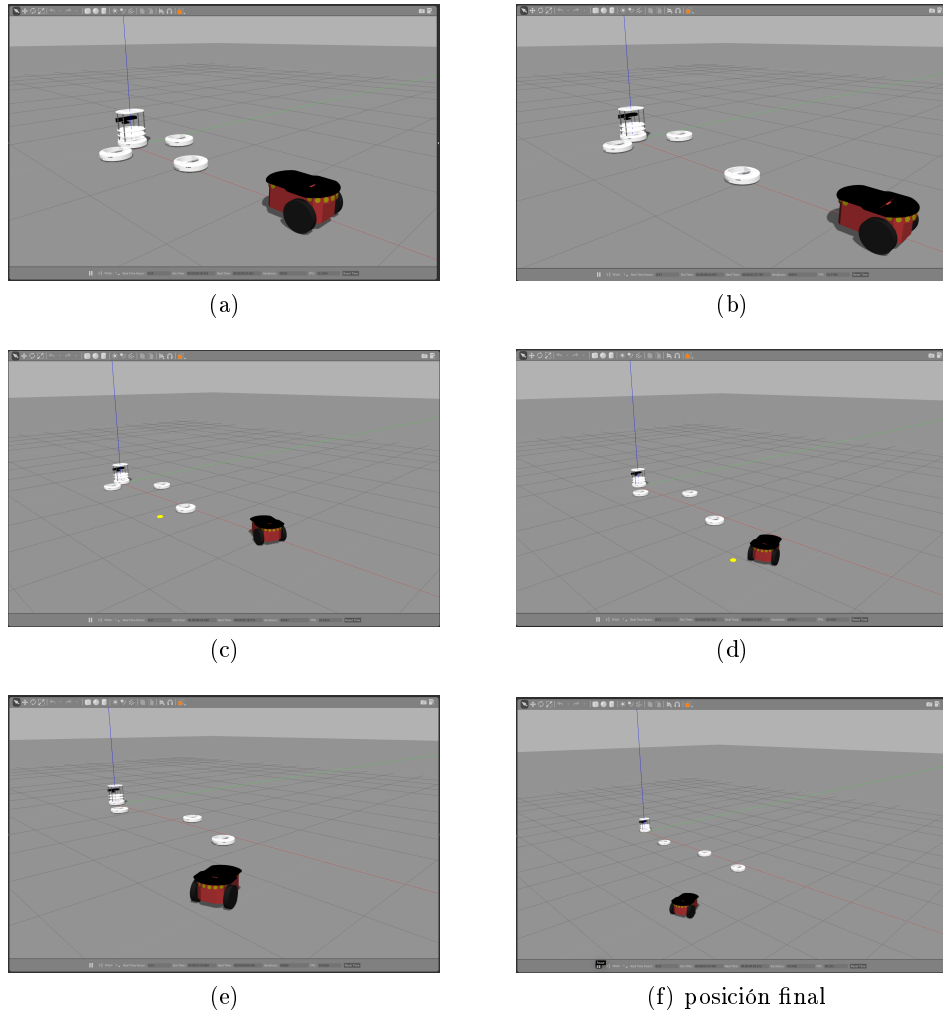


Figura 6.8: Ejecución trayectoria L derecha

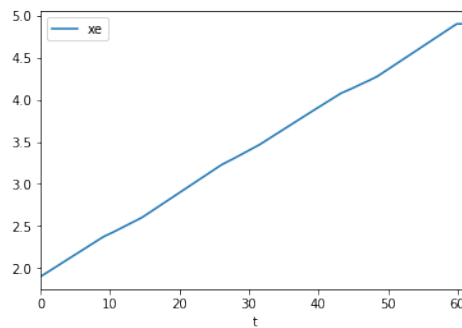
Para el segundo experimento se utilizó la trayectoria en L derecha mostrada en la Figura 6.8. El conjunto de Figuras 6.7a muestra la distribución inicial del experimento, y en la Figura 6.8, (a) muestra el despliegue del primer Robot Router Autónomo, (b) muestra el despliegue del segundo Robot Router Autónomo, (c) muestra el despliegue del tercer Robot Router Autónomo, (d) muestra la distribución cuando Robot Explorador Autónomo está en la recta final del experimento, (e) muestra la posición casi al final y (f) visualiza la distribución final de cada Robots al terminar el experimento.

6.4. Análisis de variables

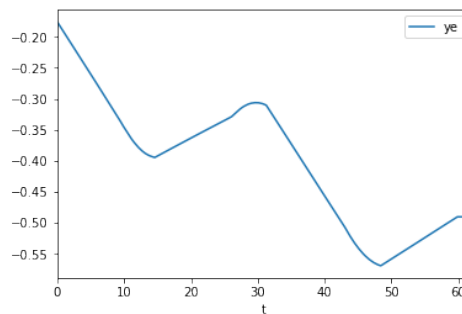
Debido a que la herramienta creada es una simulación visual, no muchas veces queda claro como van evolucionando las variables. Es por esto que se crearon gráficos al respecto que serán mostrados a continuación.

6.4.0.1. Trayectoria zigzag

Para poder entender la evolución de las variables del primer experimento(6.7) se graficó la posición del Robot Explorador Autónomo con su componente en el eje x en 6.9a y su componente en el eje y en 6.9b.

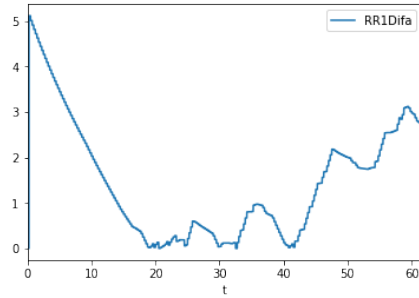


(a) posición x

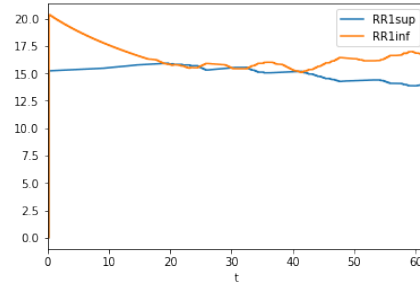


(b) posición y

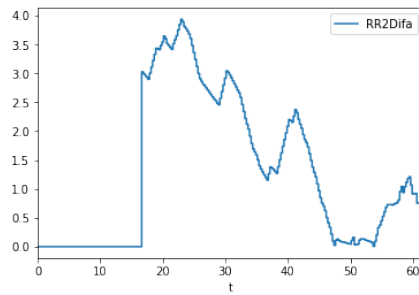
Figura 6.9: Posición Robot Explorador Autónomo en trayectoria zigzag



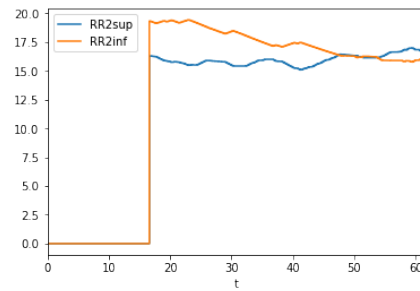
(a) Robot Router Autónomo 1



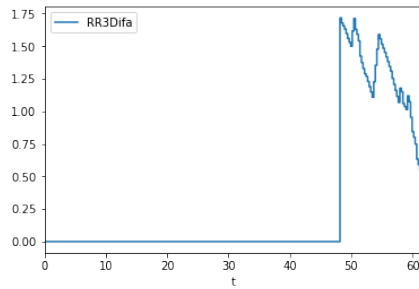
(b) Robot Router Autónomo 1



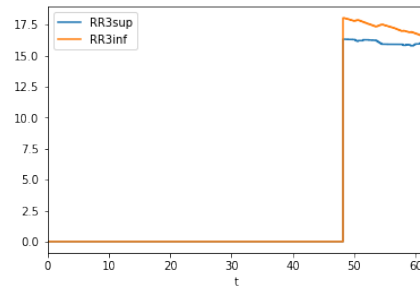
(c) Robot Router Autónomo 2



(d) Robot Router Autónomo 2



(e) Robot Router Autónomo 3

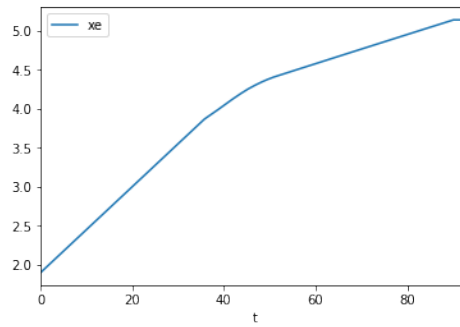


(f) Robot Router Autónomo 3

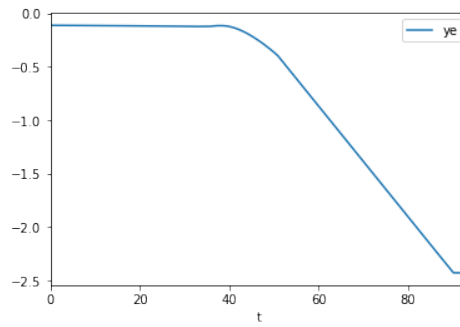
Figura 6.10: Difa, RSSIsup y RSSIinf de cada RRA en la trayectoria zigzag.

Además se graficaron los valores de $Difa$, $RSSI_{sup}$ y $RSSI_{inf}$ de cada Robot Router Autónomo mostrados en la Figura 6.10. Se puede observar que en (a), (c) y (e) logran tener una tendencia a 0 el indicador $Difa$ manteniendo una señal $RSSI_{sup}$ y $RSSI_{inf}$ buena mostrado en (b), (d) y (f).

6.4.0.2. Trayectoria L derecha



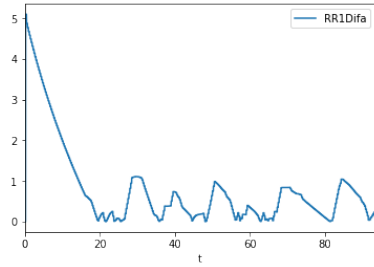
(a) posición x



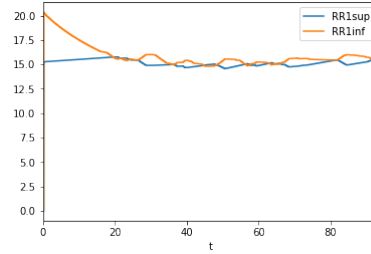
(b) posición y

Figura 6.11: Posición Robot Explorador Autónomo en trayectoria L derecha

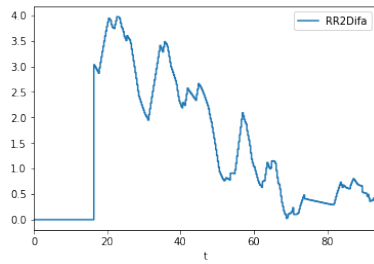
Para poder entender la evolución de las variables del segundo experimento(6.8) se graficó la posición del Robot Explorador Autónomo con su componente del eje x en 6.11a y su componente del eje y en 6.11b.



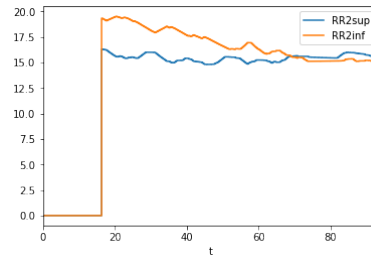
(a) Robot Router Autónomo 1



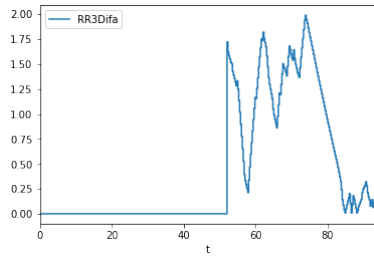
(b) Robot Router Autónomo 1



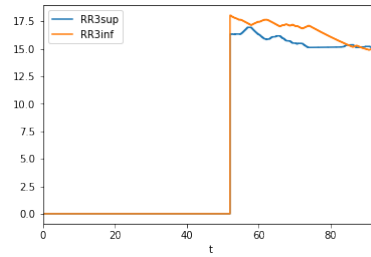
(c) Robot Router Autónomo 2



(d) Robot Router Autónomo 2



(e) Robot Router Autónomo 3



(f) Robot Router Autónomo 3

Figura 6.12: Difa, RSSI_{sup} y RSSI_{inf} de cada RRA en la trayectoria L derecha.

Para continuar con el análisis se graficaron los valores de $Difa$, $RSSI_{sup}$ y $RSSI_{inf}$ de cada Robot Router Autónomo mostrados en la Figura ???. Se puede observar que en (a), (c) y (e) logran tener una tendencia a 0 el indicador $Difa$ manteniendo una señal $RSSI_{sup}$ y $RSSI_{inf}$ buena mostrado en (b), (d) y (f).

Capítulo 7

Conclusiones

Cuando los estuديات hacen trampa en los exámenes es porque nuestro sistema escolar valora más las notas de lo que los estudiantes valoran el aprendizaje.

Neil DeGrasse Tyson

RESUMEN: En este último capítulo se dará a conocer la conclusión final sobre la simulación realizada en gazebo utilizando ROS con Cpp y algunas de las posibles líneas de continuación de este trabajo.

7.1. Conclusión

Con respecto a la estructura de programación se puede inferir que es de fácil manipulación debido a que los movimientos de los Robot Router Autónomo y Robot Explorador Autónomo son archivos de texto plano modificables y ejecutados sin problemas y sin tener la necesidad de compilar el programa.

En relación al algoritmo utilizado, al analizar los indicadores utilizados por el algoritmo de Inteligencia Artificial se llegó a la conclusión que el estado favorable de $(Min_{Dif}, Max_{vl}, Min_{vc})$ es uno entre 8 lo que provoca que el algoritmo tarde en encontrar una solución y seguir al Robot Explorador Autónomo, esto se debe al dar varios pasos aleatorios.

Como se pudo observar en el Capítulo 6, la distribución espacial inicial de los Robot Router Autónomo provoca una tendencia a quedarse en el mismo cuadrante sin importar la dirección del Robot Explorador Autónomo. Esto es debido a que el algoritmo utilizado tiene como solución óptima una recta infinita donde se intersectan los módulos de la posición del robot sucesor y el antecesor. Por otro lado, se puede observar que el factor de tiempo real varía

con tendencia descendiente debido al cálculo de todas las variables, a las comunicaciones implementadas y a la capacidad del computador utilizado, provocando pequeñas variaciones en la trayectoria del robot debido a que el tiempo de programación es diferente al tiempo de simulación. También se puede observar que en los dos experimentos los Robot Router Autónomo consiguen cumplir con mantener una buena calidad de señal RSSI e independiente de la trayectoria del Robot Explorador Autónomo .

Tomando en cuenta los resultados analizados en las figuras de análisis de monitor del sistema (Capítulo 6) se puede inferir que el código, estructuras de programación , funciones y variables utilizadas demandan un gran uso de hardware computacional.

Como conclusión general se puede decir que el programa ejecuta correctamente el algoritmo de Inteligencia Artificial seleccionado y es modificable fácilmente debido a la estructura y funciones creadas.

7.2. Trabajos futuros

Como ya está creado el esquema y funciones generales para la simulación se propone investigar, analizar y mejorar en las siguientes líneas:

7.2.1. Cambio de Robots

Modificar los robots utilizados por unos con mayores prestaciones y de fácil adquisición física.

7.2.2. Modo trayectoria

La trayectoria propuesta en el programa fue continua sin una retroalimentación de posición, sólo fue establecida la velocidad de los motores. Es por esto que se propone utilizar avance de cuadrantes de forma cartesiana con un lazo de control de posición espacial.

7.2.3. Cambio de parámetros de los robots

Para cada uno de los robots utilizados se programaron los mismos valores PID de los motores, por lo que se propone ir cambiando y proponiendo mejores estrategias de control de error modificando los valores proporcionales, integrales y derivativos de cada robot en particular.

7.2.4. Utilización de sensores simulados

Como obtener datos de los sensores de cada robot simulado es muy complejo se prefirió simular la señal de comunicación mediante el módulo de la

posición de cada robot. Es por esto que se propone investigar la forma de utilizar los sensores diseñados por los fabricantes de los robots.

7.2.5. Optimización del código

Debido a que el funcionamiento del programa es complejo, el computador se sobrecarga después de un tiempo. Se propone realizar un análisis del código propuesto y mejorar su funcionamiento.

7.2.6. HMI con visualización de estados

Finalmente, para el monitoreo completo de cada robot, se propone la creación de una interfaz máquina-hombre para visualizar en todo momento el estado de los robots sin utilizar el terminal de consola.

Parte I

Anéxos

Apéndice A

ROS

*Microsoft no está mal, sólo hacen
sistemas operativos muy cutres.*

Linus Torvalds

A.1. Introducción a ROS

El Robotics Operation System (ROS), desarrollado originalmente en 2007 por el Laboratorio de Inteligencia Artificial de Stanford (SAIL¹), mejorado por Willow Garage² en 2008 y transferido a la Open Source Robotics Foundation(OSRF) en 2013, es un framework(estructura conceptual y tecnologica de soporte definido) flexible para la escritura de un software robotico. Es una coleccion de herramientas, librerias, y convenciones que tiene por objetivo simplificar la tarea de crear complejos y robustos ambientes roboticos para las distintas plataformas y esta mantenido por la OSRF. Este sistema operativo atiende la gran dificultad de crear un robusto software robotico de proposito general, ya que, desde la perspectiva robotica, los problemas que son triviales para el ser humano frecuentemente varian enormemente entre instancias de tarea y entorno. Poder satisfacer estos requerimientos resulta imposible para un individuo, laboratorio o institucion solos. Como resultado, ROS fue construido desde cero para fomentar el desarrollo de robotica colaborativay asi distintos grupos a nivel mundial pueden contribuir con sus aportes a la comunidad. ROS es el SDK definitivo para el desarrollo de aplicaciones de robots. Proporciona una arquitectura distribuida y contiene algoritmos implementados del estado del arte, mantenidos por expertos en el campo. Es ampliamente adoptado por la investigacion, centros educativos y la industria. Al ser un sistema operativo de codigo abierto mantenido, utilizado y desarrollado por numerosos usuarios y entidades, se trata de una

¹<http://ai.stanford.edu/>

²<http://www.willowgarage.com/>

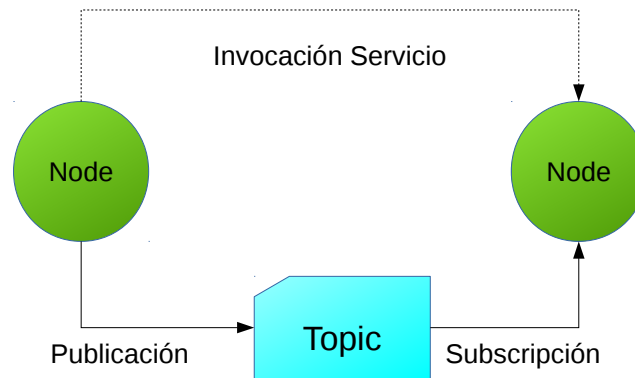


Figura A.1: Diagrama de comunicación Nodo a Nodo

infraestructura muy flexible y adaptable a diversas necesidades.

A.2. Conceptos básicos

- **Nodos:** Son los procesos que realizan el calculo. ROS esta diseñado para ser modular en una escala de grano fino; un sistema de contrl de robots comprende por lo general muchos nodos. Por ejemplo, un nodo controla un laser, un nodo contrla los motores de las ruedas, un nodo proporciona una vista gradica del sistema, y asi sucesivamente. Un nodo de ROS se escribe con el uso de las librerias cliente de ROS, `roscpp` y `rospy`.
- **Master:** Proporciona la informacion necesaria para que todos los nodos sean identificados y se comuniquen, es decir, porporciona el registro de nombre y consulta al resto del grafo de computacion. Sin el maestro, los nodos no serian capaces de encontrar al resto de nodos, intercambiar mensajes o invocar servicios.
- **Parameter Server:** Forma parte del Master, y permite guardar los datos de forma centralizada con una clave en una locacion central.
- **Messages:** Estos mensajes son usados por los nodos para que puedan comunicarse entre ellos. Es una estructura de datos compuestos que comprende una combinacion de tipos primitivos y mensajes.

- Primitivos: Los tipos primitivos de datos (integer, floating point, boolean, etc.) están soportados como los arrays de tipo primitivo.
 - Mensajes: Los mensajes pueden incluir estructuras y arrays (como las estructuras de C).
- Topic: Es un nombre que es usado para identificar el contenido del mensaje. Los mensajes son enrutados por un sistema de transporte que utiliza semántica publicador/subscriptor. Un nodo envía mensajes publicando en un tópico. Un nodo que está interesado en cierto tipo de datos se suscribirá al tópico apropiado. Pueden existir muchos publicadores concurrentemente para un solo tópico. En general, los publicadores y subscriptores no son conscientes de la existencia de los otros. La idea es desacoplar la producción de información de su consumo. Lógicamente, uno puede pensar que un tópico es un mensaje fuertemente tipado³ en un bus. Ese bus tiene un nombre y nadie se puede conectar al bus para enviar o recibir mensajes si no están correctamente tipados.
 - Services: Es el modelo de comunicación cliente-servidor. El modelo de publicar/subscriptor es un paradigma de comunicación muy flexible, pero en muchos casos el transporte en un único sentido no es suficiente para las interacciones de petición y respuesta que a menudo se requieren en un sistema distribuido. La petición y respuesta se realiza a través de los servicios, que se definen a partir de una estructura de mensajes: una para la petición y otra para la respuesta. Un nodo proporciona un servicio con un nombre y un cliente, utiliza dicho servicio mediante el envío del mensaje de petición y espera a la respuesta. La librería cliente de ROS generalmente presenta esta interacción como si fuera una llamada a procedimiento remoto.
 - Bags: Es un formato para guardar y reproducir datos de mensajes ROS. Son un mecanismo importante para el almacenamiento de datos, como lecturas de sensores, que pueden ser difícilmente adquiridas pero son necesarias para el desarrollo y testeado de algoritmos.

Se puede observar un esquema gráfico en la Figura A.1

³no se permiten violaciones de los tipos de datos, es decir, dado el valor de una variable de un tipo concreto, no se puede usar como si fuera de otro tipo distinto a menos que se haga una conversión.

A.3. Componentes Principales

A.3.1. Infraestructura de comunicaciones

Un sistema de comunicación es primordial para la implementación de una aplicación robótica. El sistema de mensajería integrado de ROS permite ahorrar tiempo mediante la gestión de los detalles de la comunicación entre los nodos distribuidos a través de publicación anónima/método de suscripción, obligando a implementar interfaces claras entre los nodos del sistema, mejorando así la encapsulación y la promoción de la reutilización de código. La estructura de estas interfaces de mensaje se define en el mensaje de IDL(Interface Description Language).

A.3.2. Características Robóticas específicas

ROS proporciona librerías robóticas específicas comunes y herramientas para que el robot funcione rápidamente. A continuación se describirán algunas de las características robóticas específicas ofrecidas por ROS.

A.3.2.1. Mensajes estándar de robot

Años de discusión y desarrollo de la comunidad han dado lugar a un conjunto de formatos de mensajería estándar que cubren la mayor parte de los casos de uso común en la robótica. Hay definiciones de mensajes de conceptos geométricos como poses, transformadas y vectores; para sensores como cámaras, IMU y láser; y para los datos de navegación como odometría, rutas y mapas; entre muchos otros. Estos mensajes también incrementan la interoperabilidad con las herramientas y características existentes en el ecosistema de ROS.

A.3.2.2. Librería geométrica de robot

Un reto común en muchos proyectos de robótica es hacer el seguimiento de distintas partes del robot y dónde se localizan con respecto a la otra. Por ejemplo, si desea combinar datos de una cámara con los datos de un láser, lo que necesita saber dónde está cada sensor, en algún marco de referencia común. Por esta razón se utilizará la librería tf (transformada), que mantiene el seguimiento de todas las referencias del sistema robótico. Diseñado para maximizar la eficacia, la librería tf se ha utilizado para administrar las transformadas de los datos para robots con más de un centenar de grados de libertad (figura A.2) y las tasas de actualización de cientos de Hertzios. La librería tf permite definir las transformaciones estáticas, tales como una cámara que se fija a una base móvil, y las transformaciones dinámicas, como una articulación de un brazo robótico.

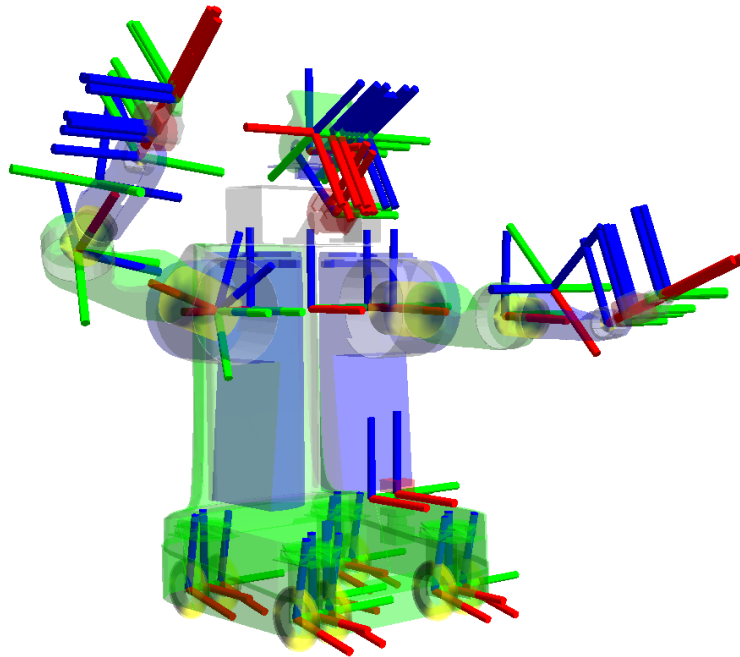


Figura A.2: Geometría de Robots en ROS

A.3.2.3. Unidades Utilizadas

Para poder trabajar con ros de manera general y global se utiliza el siguiente sistema internaciona de medidas mantenidos por la Bureau International des Poids et Mesures ⁴, las cuales se muestran a continuación.

Cantidad	Unidad
longitud	metro
masa	kilogramo
tiempo	segundo
corriente	Amper

Tabla A.1: Unidades Básicas

A.3.2.4. Estructura del Robot

ROS proporciona un conjunto de herramientas para describir y modelar al robot para que pueda ser comprendido por el resto de su sistema, incluyendo `tf`, `robot_state_publisher`, `rviz`, `gazebo` o cualquier plugins o software compatible. El formato para la descripción de su robot de ROS es URDF (Unified Robot Descripción Format), que consiste en un documento XML

⁴<http://www.bipm.org/en/home/>

Cantidad	Unidad
angulo	radian
frecuencia	Hertz
fuerza	Newton
potencia	Watt
voltaje	Volt
temperatura	Celsius
magnetismo	Tesla

Tabla A.2: Unidades Derivadas

en el que se describen las propiedades físicas de su robot, desde la longitud de las extremidades y tamaños de ruedas a la ubicación de los sensores y la apariencia visual de cada parte del robot. Una vez que se define de esta manera, el robot se puede utilizar fácilmente con la librería `tf`, representada en las tres dimensiones con visualizaciones detalladas, y se utiliza con los simuladores y los planificadores de movimiento. Es por esto que surge la necesidad de definir algunos conceptos como(ver 4.2):

- **Link:** Código que describe la física de cada estructura del robot, tales como la matriz inercial, la visualización, la colisión, etc.
- **Joint:** También llamado juntura, es el código encargado de determinar la relación entre los links y los límites de sus comportamientos, como por ejemplo la velocidad máxima y mínima de movimiento, las constantes de posición y velocidad del motor que determina su curva de movimiento, entre otras.
- **Parent:** Es el link base o maestro determinado por el joint.
- **Child:** Es el link esclavo determinado por el joint.

A.3.2.5. Diagnósticos

El sistema de diagnóstico es diseñado para recoger la información desde controladores de hardware y el hardware del robot a usuarios y operadores para el análisis, solución y logging. El repertorio de diagnóstico contiene herramientas para coleccionar, publicar, analizar y visualizar datos de diagnóstico. El diagnóstico toolchain es construido en relación al topic `/diagnostic`. Sobre este topic, los controladores de hardware y dispositivos publican mensajes `diagnostic_msgs/DiagnosticArray` con los nombres de dispositivos, el estado y puntos de datos específicos. Los paquetes `diagnostic_updater` y `self_test` permiten a los nodos recoger y publicar datos de diagnóstico. El `diagnostic_aggregator` puede clasificar y analizar el diagnóstico en el tiempo de ejecución. Los operadores y desarrolladores pueden ver los datos de diagnóstico utilizando el paquete de `rqt_robot_monitor`. El paquete

`diagnostic_analysis` puede convertir el registro de diagnósticos a un archivo CSV para examinar y realizar un posterior análisis.

A.3.3. Herramientas

una de las características de ROS es el poderoso conjunto de herramientas de desarrollo que incluye. Estas ayudan con la introspección, la depuración, el trazado, y visualización del estado en el desarrollo del sistema las cuales se aprovechan a través de una extensa colección de utilidades de línea de comandos y gráficos que simplifican el desarrollo y depuración. Las más utilizadas se mostrarán a continuación.

A.3.3.1. Rosnode

- `rostopic info node`: Muestra información sobre el nodo.
- `rostopic kill node`: Mata el nodo o envía una señal para matarlo.
- `rostopic list`: Muestra una lista con los nodos activos.
- `rostopic machine hostname`: Lista los nodos ejecutándose en una máquina en concreto.
- `rostopic ping node`: Muestra la conectividad con el nodo.
- `rostopic cleanup`: Limpia la información de registro para nodos inalcanzables.

A.3.3.2. Rostopic

- `rostopic bw /topic`: Muestra el ancho de banda utilizado por un tópico.
- `rostopic echo /topic`: Muestra el mensaje por la salida estándar.
- `rostopic find message_type`: Busca tópicos que usen el tipo de mensaje especificado.
- `rostopic hz /topic`: Publica la tasa de publicación del tópico.
- `rostopic info /topic`: Muestra información sobre el tópico, el tópico publicado, los que están suscritos y los servicios.
- `rostopic list`: Muestra información sobre los tópicos activos.
- `rostopic pub /topic type args`: Publica datos al tópico. Permite crear y publicar datos en cualquier tópico directamente desde la línea de comandos.

- `rostopic type /topic`: Muestra el tipo de topico, es decir, el tipo de mensaje que publica.

A.3.3.3. Roservice

- `rosservice call /service args`: Llama al servicio con los argumentos apropiados.
- `rosservice find msg-type`: Busca los servicios por el tipo de servicio.
- `rosservice info /service`: Muestra la informacion sobre el servicio.
- `rosservice list`: Lista los servicios activos.
- `rosservice type /service`: Muestra el tipo de servicio.
- `rosservice uri /service`: Muestra el servicio ROSRPC URI.

A.3.3.4. Rosparam

- `rosparam list`: Lista todos los parametros del servidor.
- `rosparam get parameter`: Devuelve el valor de un parametro.
- `rosparam set parameter value`: Establece el valor de un parametro.
- `rosparam delete parameter`: Elimina un parametro.
- `rosparam dump file`: Guarda los parametros de servidor de parametros.
- `rosparam load file`: Carga un fichero con parametros en el servidor de paramatros.

A.3.3.5. Package

- `rospack`: Este comando se usa para obtener informacion o buscar informacion sobre un paquete.
- `roscreeate-pkg`: Crear un nuevo paquete.
- `rosmake`: Compilar un paquete.
- `rosdep`: Instalar en el sistema las dependencias de un paquete.
- `rxdeps`: Ver las dependencias de un paquete como un grafo.

A.3.3.6. Rosbash

- **roscd**: Cambiar de directorio.
- **roscd**: Editar un archivo.
- **roscp**: Copiar un fichero de algun paquete.
- **rosls**: Listar los directorios de un paquete.
- **rosls**: Listar los ficheros de un paquete.

A.3.4. Integración con librerías

- **Gazebo**: Gazebo es un simulador de entornos 3D que posibilita evaluar el comportamiento de un robot en un mundo virtual. Permite, entre muchas otras opciones, diseñar robots de forma personalizada, crear mundos virtuales usando sencillas herramientas CAD e importar modelos ya creados. Además, es posible sincronizarlo con ROS de forma que los robots emulados publiquen la información de sus sensores en nodos, así como implementar una lógica y un control que dé ordenes al robot.
- **Moveit**: Librería de planificación de movimientos que ofrece implementaciones eficientes y bien probados de los algoritmos de planificación que se han utilizado en una amplia variedad de robots, desde plataformas con ruedas sencillas de humanoides para caminar.
- **Rviz**: proporciona la visualización tridimensional de muchos tipos de datos de los sensores y cualquier robot descrito por URDF. La herramienta rviz puede visualizar muchos de los tipos de mensajes comunes previstos en ROS, como escaneos láser, las nubes de puntos tridimensionales, y las imágenes de cámaras.
- **V-rep**: Simulador 3D que permite diferentes modos de simulación y trae unos algoritmos a primera vista bastante atractivos para el cálculo de elementos esenciales en cualquier robot como son la cinemática inversa y la detección de colisiones.
- **Blender**: Software Open-Source destinado, en primera instancia, al modelado 3D de objetos. Incorpora la posibilidad de dar texturas, definir materiales, iluminar la escena, crear simulaciones... Destaca por sus capacidades para la creación de mallas y animaciones.

Apéndice B

Gazebo

*La computacion es el principio, el
ordenador es la herramienta*

Robert Denning

RESUMEN: En este capítulo veremos el poderoso simulador Gazebo desde los componentes principales hasta algunos de los comandos mas útiles.

B.1. Introducción a Gazebo

Gazebo es un simulador 3D multi-robot, con propiedades dinámicas y cinemáticas completas. La integración entre ROS y Gazebo es proporcionada por un conjunto de plugins de Gazebo que apoyan muchos robots y sensores existentes. Debido a que los plugins presentan la misma interfaz de mensaje que el resto del ecosistema ROS, se pueden escribir nodos ROS que son compatibles con la simulación, los datos registrados y el hardware de los robots.

B.2. Componentes Principales

B.3. Ejecución

Gazebo puede ejecutarse como simulador stand-alone, pero nosotros lo usaremos conjuntamente con ROS para poder consultar desde los nodos que creemos la información de los sensores, así como para mandarles comandos de velocidad. Para lanzar el simulador primero tenemos que arrancar en un terminal el motor de ROS:

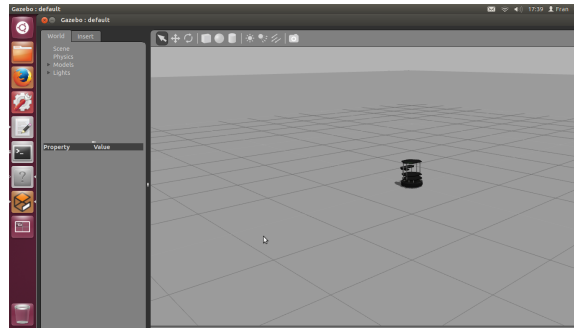


Figura B.1: Vista del mundo simulado por gazebo con un robot turtlebot

`roscore`

En otra terminal lanzamos el siguiente comando:

```
roslaunch turtlebot_gazebo turtlebot_empty_world.launch
```

`roslaunch` es una herramienta que sirve para lanzar varios nodos de ROS de un paquete de forma automática y con una configuración específica impuesta por un fichero XML con extensión `.launch`.// Tras ejecutar el comando se abrirá la ventana de Gazebo con un robot cargado situado en el centro del mundo (figura B.1)

B.4. Utilidades

Debido a que en esta sección se da a conocer las utilidades del simulador Gazebo tomaremos como ejemplo el uso del robot Turtlebot para facilitar el entendimiento de las funcionalidades de gazebo junto a ROS.

B.4.1. Control de cámara

El control del punto de vista de diseño es parecido al que muestran otros programas de diseño o simulación 3D:

- El botón izquierdo del ratón sirve para trasladar la cámara por el mundo
- El botón derecho o la rueda del ratón sirve para hacer zoom en la escena
- El botón central del ratón sirve para rotar la cámara

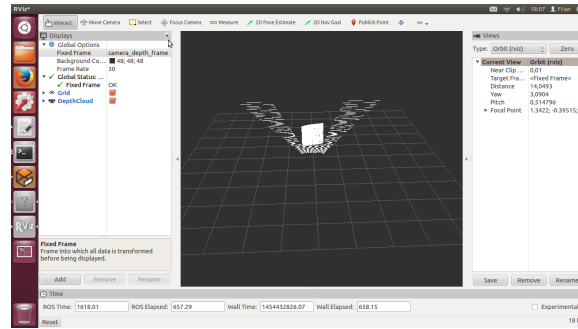


Figura B.2: Visualización en Rviz de la obtención de datos del robot Turtlebot

B.4.2. Diseño de Mundos

Gazebo permite diseñar elementos con herramientas CAD, dichas herramientas son muy limitadas, pero suficientes para crear mundos sencillos. Estas herramientas están situadas en la zona superior del programa y nos permite insertar cubos, esferas y cilindros, así como distintos puntos de luz. Las otras tres herramientas sirven para seleccionar elementos, trasladarlos y rotarlos. Los elementos que contiene el mundo se van listando en la parte izquierda del programa, bajo la pestaña "Worlds" si son seleccionados permiten la edición de algunas propiedades tales como la pose o el nombre.

B.4.3. Visualización de Sensores

Por el hecho de tener ROS funcionando y haber usado roslaunch para lanzar el simulador, ya tenemos publicada la información que captan los sensores en la simulación. Para visualizarla usaremos otra herramienta de ROS llamada RViz, por lo que ejecutamos en otro terminal:

```
roslaunch rviz rviz
```

El comando lanza el programa pero aparentemente no muestra información alguna. Esto es porque aún no estamos suscritos a ningún tópico de los del robot. Antes de suscribirnos a un tópico, vamos a configurar el marco de referencia global del visualizador, que por defecto está en `map`, y lo cambiamos a `odometry`. El marco fijo hace referencia a respecto a qué se va a mostrar la información en el visualizador. Si elegimos la opción `camera_link` la información se representará en base a la posición de la articulación de la cámara. Si elegimos la opción `camera_depth_frame` la información se representará en base a la posición del sensor de profundidad. Lo ideal es que el marco de referencia global haga referencia a un elemento que es fijo. Ahora sí, para suscribirnos a un nodo pulsamos en el botón "Add" de la parte in-

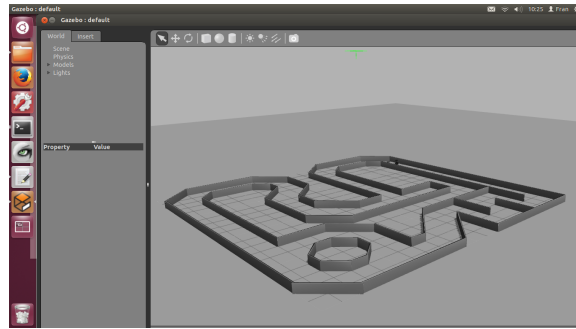


Figura B.3: Visualización del mundo creado con dos robots

inferior izquierda. En la nueva ventana que se abre seleccionamos la pestaña "By topic" elegimos uno de ellos en función de la información que queramos visualizar, por ejemplo `camera/depth/image_raw/DepthCloud` y pulsamos ".OK". El visualizador mostrará la nube de puntos captada por la cámara RGB-D, que si has seguido los pasos consistirá en un cono de puntos a ras del suelo y una pared formada por uno de los lados del cubo el resultado se puede observar en la figura B.2.

B.4.4. Importación de modelos 3D al mundo

Lo siguiente que vamos a hacer es importar un fichero de configuración que carga el robot en un mundo que hemos diseñado con anterioridad. Como se ha explicado arriba, el paquete que necesitamos es el de `turtlebot_gazebo` por lo que necesitamos ubicarlo. Para ello usamos el comando `rospack`

```
rospack find turtlebot_gazebo
```

Lo siguiente es crear un nuevo paquete de catkin en nuestro workspace y copiar dentro todo el contenido del paquete `gazebo_turtlebot`. Seguiremos cambiándole el nombre del proyecto en el fichero `package.xml` para que cuadre con el nombre del paquete que hemos creado. Por último, copiamos dentro de la carpeta `launch` los ficheros que se proporcionan a continuación `turtlebot_gazebo_multiple_files.tar.gz`. Estos ficheros contienen la configuración del simulador y de algunos nodos que se lanzan conjuntamente.

El primer fichero `create_multi_robot.launch` especifica la configuración del mundo que se va a usar. En este caso se toma como base un mundo vacío en el que se inserta el diseño del mundo, especificado en otro fichero que se proporciona `race.sdf`. En el siguiente bloque se incluye la configuración de los robots, que está definida en `robots.launch`. Aprovechamos para cambiar las rutas del paquete por el nombre del paquete que hayamos creado en el comando `find`. En `robots.launch` se especifica la configuración de los dos robots

que vamos a cargar en el simulador. En el primer bloque se definen ciertas variables como la base del robot, el conjunto de "bandejas" que forma el cuerpo del turtlebot o el tipo de cámara que monta. Por ejemplo, podemos ver que la base que usará el robot se llama kabuki, que va a usar las plataformas hexagonales, en vez de las circulares y que va a montar una cámara kinect. El siguiente bloque carga el modelo del robot, que es el mismo para los dos. Luego hay dos bloques que definen dos robots usando el modelo anterior, pero especificando algunos parámetros únicos como el nombre o la pose inicial. En esta especificación se hace referencia al último fichero `one_robot.launch` que se encarga de hacer aparecer el robot en el simulador y de cargar otros nodos relacionados con la redirección de los tópicos del robot como, por ejemplo, `robot_state_publisher` que se va a encargar de exponer todos los nodos que tiene el robot, con la información de las cámaras, odometría, láser y otros. También se especifica un nodo que simula la salida de un láser a partir de la imagen de profundidad obtenida por kinect. Cambiamos las rutas y los nombres de los paquetes por los correctos. El otro fichero que se proporciona, `race.sdf`, contiene la definición de los elementos que forman el mundo. En este caso es un mundo formado por un sol, que nos proporciona un sistema de iluminación, un plano que hace de suelo y varios elementos con geometría tipo caja que forman el mundo. Además de las dimensiones también se especifican la pose (traslación y rotación con respecto del sistema de referencia global), algunas propiedades físicas como la inercia o la masa y otras relacionadas con la textura que va a mostrar. Una vez explicado todo esto, procedemos a construir el paquete y a cargar el mundo y los turtlebot en el simulador con el siguiente comando:

```
catkin_make
source devel/setup.bash
roslaunch turtlebot_gazebo_multiple create_multi_robot.launch
```

Hecho lo anterior se puede observar en la figura B.3 el mundo que consta de una serie de paredes modeladas a partir de las cajas que se veían en el fichero `race.sdf` ubicadas de tal manera que forman un circuito y dos turtlebots situados en la línea de salida.

B.4.5. Comandos Útiles

B.4.5.1. Movimiento de un robot

La forma de mover los robots, tanto en el mundo real como en el simulador, es enviándoles comandos de velocidad. Los robots tienen un nodo que consumen mensajes de tipo `geometry_msgs/Twist` que son el tipo de mensajes para indicar precisamente eso, valores para velocidades lineales y angulares. Un mensaje de este tipo acepta como máximo 6 valores: las velocidades lineales en x, y, z y las velocidades angulares de roll, pitch y yaw.

Estos valores permiten dar movimiento a robots con hasta 6 grados de libertad, pero turtlebot sólo puede moverse por el plano XY y rotar en el eje Z (yaw). Pero antes de enviar nada es necesario averiguar qué tópico es que acepta este tipo de mensajes por lo que usamos el siguiente comando para ver qué tópicos tenemos en el sistema:

```
rostopic list
```

Para obtener más información sobre algún tópico usamos el siguiente comando

```
rostopic info /robot1/commands/velocity
```

El comando para empezar a mover el robot es el siguiente:

```
rostopic pub -r 10 /robot1/commands/velocity geometry_msgs/Twist '{linear: {x: 0.1}}'
```

Este comando sirve para obtener información de los nodos, mostrar su feedback y para enviarles mensajes. Nosotros usaremos el modo pub (publish) para publicar mensajes de velocidad. El parámetro -r indica la frecuencia en que se envían los mensajes, en este caso 10Hz. El siguiente argumento indica el tópico donde será lanzado este mensaje. El siguiente hace referencia al tipo de mensaje, en este caso uno de tipo Twist. Y por último se indica el contenido del mensaje, en este caso velocidad lineal de 0.1 unidades (m/s).

Bibliografía

*Y así, del mucho leer y del poco dormir,
se le secó el cerebro de manera que vino
a perder el juicio.*

Miguel de Cervantes Saavedra

- [1] CAPARRINI, F. S. Aprendizaje por refuerzo: algoritmo q learning. 2 Marzo 2019. Disponible en <http://www.cs.us.es/~fsancho/?e=109> (último acceso, Abril, 2019).
- [2] CORPORATION, I. Irobot create. 2002. Disponible en <https://robots.ros.org/irobot-roomba/> (último acceso, Abril, 2019).
- [3] DORIGO, M., ROOSEVELT, A. F. D. y DORIGO, M. Swarm robotics. *Special Issue?, Autonomous Robots*, páginas 111–113, 2004.
- [4] DORIGO, M., TUCI, E., GROSS, R., TRIANNI, V., LABELLA, T. H., NOUYAN, S., AMPATZIS, C., DENEUBOURG, J.-L., BALDASSARRE, G., NOLFI, S., MONDADA, F., FLOREANO, D. y GAMBARDELLA, L. M. The swarm-bots project. página 3342, 2005.
- [5] MOLINA, P. Historia de la robótica. Unkown. Disponible en <http://www.profesormolina.com.ar/tecnologia/robotica/historia.htm> (último acceso, Abril, 2019).
- [6] OLATE, J. M. P. y DURAN-FAUNDEZ, C. *On Maintaining Connectivity of a Colony of Autonomous Explorer Mobile Robots*. 2014.
- [7] PEZESHKIAN, N. J. D., N. y HART. *Link quality estimator for a mobile robot. Technical report, DTIC Document.*. A., 2012.
- [8] PIONEER. Pioneer 2dx. 2016. Disponible en http://wiki.ros.org/action/show/Robots/AMR_Pioneer_Compatible?action=show&redirect=Robots%2FPioneer (último acceso, Abril, 2019).
- [9] ŞAHIN, E. Swarm robotics: From sources of inspiration to domains of application. páginas 10–20, 2005.

-
- [10] TEKDas, I. V. L. J. H., O. y TERZIS. *Using mobile robots to harvest data from sensor fields. Wireless Communications, IEEE*, 16(1):22-28. A., 2009.

*—¿Qué te parece desto, Sancho? — Dijo Don Quijote —
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.*

*Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes*

*—Buena está — dijo Sancho —; firmela vuestra merced.
—No es menester firmarla — dijo Don Quijote—,
sino solamente poner mi rúbrica.*

*Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes*

