TRABAJO FIN DE GRADO

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

"Modelado, control y optimización de un sistema multi-robot autónomo para transporte inteligente"

IN LUMINE · UNIVERSITAS

Curso 2019/2020

Alumno/a:

Álvaro Ramajo Ballester

Director/es:

José Carlos Moreno Úbeda

Agradecimientos

Al Director José Carlos Moreno Úbeda, por su gran dedicación y compromiso con el proyecto.

Al Departamento de Informática, por facilitarme cuanto ha sido necesario para llevar a cabo este trabajo.

A todo el profesorado que ha participado en mi formación a lo largo de la carrera.

Dedicatoria

A toda mi familia, y en especial a mis padres,
por ser un apoyo incondicional desde que tengo uso de razón.

Índice general

Agradecimientos I

Dedicatoria III

Índice general V

Índice de figuras IX

Índice de tablas XIII

Resumen XV

Abstract XVII

1. Introducción 1

1.1. Interés y motivación del Trabajo Fin de Grado 1
1.2. Objetivos 2
1.3. Contexto 3
1.4. Resumen de resultados 6
1.5. Planificación temporal 8
1.6. Competencias utilizadas en el TFG 8
1.7. Estructura del Trabajo Fin de Grado 12

2. Estado del arte 13

2.1. Simultaneous Localization And Mapping (SLAM) 13
2.2. Localización del robot 17
2.3. Navegación autónoma 20
2.3.1. Planificación de la navegación autónoma 20
2.3.2. Arquitecturas de control 22
2.3.3. Algoritmos de planificación de rutas 24
2.4. Sistemas multi-robot 29

3. Materiales y métodos **33**

3.1. Arquitectura hardware del robot 33
3.1.1. iRobot® Create® 2 33
3.1.2. Raspberry Pi Model 3B+ 37
3.1.3. Slamtec RPLIDAR A3M1 38
3.1.4. Batería LiPo 11,1V 41
3.1.5. Convertidor buck DC-DC 41
3.2. Arquitectura software del robot 42
3.2.1. Sistema Operativo 42
3.2.2. Robot Operating System (ROS) 44
3.2.2.1. Historia 44
3.2.2.2. Uso de ROS 47
3.2.3. Gazebo 49
3.2.4. Interfaz con los dispositivos 50
3.2.4.1. iRobot ® Create ® 2 50
3.2.4.2. RPLIDAR A3M1 51
3.2.5. Software de navegación 51
3.2.5.1. SLAM 51
3.2.5.2. Localización 51

3.2.5.3. Navegación autónoma . . . 52
3.2.5.4. Planificación multi-robot . . . 53

4. Resultados **55**
4.1. Arquitectura hardware del prototipo final . . . 56
4.2. Arquitectura software propuesta . . . 57
4.3. Modelado del robot en ROS y Gazebo . . . 59
4.4. Mapeo y localización simultáneos (SLAM) en simulación . . . 69
4.5. Mapeo y localización simultáneos (SLAM) real . . . 75
4.6. Navegación autónoma en simulación . . . 77
4.7. Navegación autónoma real . . . 85
4.8. Navegación multi-robot en simulación . . . 86
4.9. Navegación multi-robot real . . . 88
4.10. Planificación multi-robot . . . 90
4.11. Planificación multi-robot real . . . 99

5. Conclusiones y trabajos futuros **103**

Anexos **105**
A. Presupuesto del proyecto . . . 105
B. Presupuesto comercial . . . 105
C. Código fuente . . . 106
C.1. Paquete tfg_simulations . . . 106
C.2. Paquete tfg_roomba . . . 152
C.3. Paquete tfg_remote . . . 160

Bibliografía **162**

Índice de figuras

1.3.1. Instalación anual de robots por industrias (2016 - 2018) 4
1.3.2. Instalación anual de robots industriales en el mundo (2016 - 2018) . . 4
1.3.3. Robots de servicio vendidas para uso personal / doméstico. Unidades vendidas en 2017 y 2018, previsión para 2019-2022 5
1.3.4. Robots de servicio vendidas para uso profesional. Unidades vendidas en 2017 y 2018, previsión para 2019-2022 5
1.4.1. Comparativa de tiempo invertido por el sistema real 7
1.4.2. Comparativa de distancia recorrida por el sistema real 7
1.4.3. Comparativa de tiempo invertido por el sistema en simulación 7
1.4.4. Comparativa de distancia recorrida por el sistema en simulación . . . 8

2.1.1. SLAM como un *factor graph*. (Fuente: Cadena et al., 2016) 16
2.3.1. Esquema de navegación reactiva 22
2.3.2. Esquema de navegación deliberativa 23
2.3.3. Esquema de navegación híbrida 24

3.1.1.1. Dimensiones iRobot Create 2 34
3.1.1.2. Sensores infrarrojos horizontales (superior) y verticales (inferior) . . . 34
3.1.1.3. Sensores de contacto 35
3.1.1.4. Sensores de caída de rueda 35
3.1.1.5. Sensores infrarrojos base - robot 35

3.1.1.6. Actuadores 36
3.1.2.1. Raspberry Pi Model 3B+ 38
3.1.3.1. Slamtec RPLIDAR A3M1 39
3.1.3.2. Especificaciones RPLIDAR A3M1 40
3.1.3.3. Triangulación láser RPLIDAR A3M1 40
3.1.3.4. Ventana óptica RPLIDAR A3M1 40
3.1.4.1. Batería LiPo U-TECH 2200 mAh 41
3.1.5.1. Convertidor buck DC-DC 42
3.2.1.1. Nivel de abstracción del sistema operativo 43
3.2.2.2.1. Versiones ROS 47
3.2.5.3.1. Diagrama del *stack* de navegación de ROS 52
3.2.5.3.2. Diagrama de los movimientos de recuperación 52
3.2.5.3.3. Diagrama de *nav_core* 53

4.1.1. Arquitectura hardware 56
4.1.2. Prototipo final 56
4.1.3. Prototipo final (vista frontal) 57
4.2.1. Arquitectura software 58
4.3.1. Errores del modelo previo 59
4.3.2. Modelo final 66
4.3.3. Comparativa de modelos (visual y de colisiones) 67
4.4.1. Proceso de mapeo 75
4.4.2. Mapa construido (izquierda) y mapa real (derecha) 75
4.5.1. Mapa real 77
4.6.1. Coste según el radio de inflado 81
4.6.2. Mapa real (SLAM) 84
4.6.3. Entorno virtual 84

4.6.4. Visualización del la navegación en entorno virtual 85
4.6.5. Comparativa de tiempos de navegación 86
4.8.1. Cruce frontal 87
4.8.2. Cruce a noventa grados 87
4.8.3. Cruce frontal de cuatro robots (inicio) 88
4.8.4. Cruce frontal de cuatro robots (final) 88
4.9.1. Mapa nuevo 89
4.9.2. Cruce frontal RVIZ (1) 89
4.9.3. Cruce frontal real (1) 89
4.9.4. Cruce frontal RVIZ (2) 90
4.9.5. Cruce frontal real (2) 90
4.10.1. Planificación multi-robot en simulación (1) 97
4.10.2. Planificación multi-robot en simulación (2) 97
4.10.3. Planificación multi-robot en invernadero 97
4.10.4. Comparativa de tiempos en simulación 98
4.10.5. Comparativa de distancia recorrida en simulación 98
4.11.1. Estructura de nodos del sistema multi-robot 99
4.11.2. Puntos de recogida 100
4.11.3. Puntos de entrega 100
4.11.4. Comparativa de tiempos 101
4.11.5. Comparativa de distancia recorrida 101

Índice de tablas

1.5.1. Cronograma temporal 9

2.2.1. Pseudocódigo del algoritmo MCL(X, a, o) 19

A.1. Presupuesto del proyecto 105

Resumen

La robótica es el área de la ingeniería dedicada al diseño, construcción y control de máquinas capaces de resolver problemas a los humanos. Es una rama interdisciplinar que, aunando los conocimientos de diversas ciencias, ingenierías y tecnologías, busca imitar o extender las acciones y capacidades de las personas.

Son incontables las labores en las que los robots demuestran un desempeño mayor que los humanos, como bien es consciente el sector industrial. Su velocidad, precisión y repetibilidad en tareas monótonas están un orden de magnitud por encima de las habilidades humanas. Sin embargo, la habilidad de establecer vínculos entre ellos, coordinarse e integrarse en su entorno de manera eficaz sigue siendo una tarea pendiente.

Con esta finalidad, el presente trabajo expone los problemas diversos e interrelacionados entre sí que se deben abordar para conseguir aunar los esfuerzos de múltiples robots para conseguir un objetivo común. Estos retos comienzan con la percepción del entorno y la manera en la que se modela. Mediante dispositivos tipo lidar y el mapeado y localización simultáneos (SLAM) se consigue una descripción fiel del mundo que rodea al robot. Aún así, también se manifiesta en el trabajo las limitaciones que sufren estos dispositivos.

La construcción de un mapa es la base sobre la que se asienta una navegación autónoma eficiente. Los algoritmos de navegación continúan estudiándose desde los inicios de la robótica móvil, sin embargo, cada uno de ellos ofrece un enfoque distinto que se adecua a cada caso. Por ello, se han llevado a cabo pruebas para comprobar cuales permiten desplazarse en un menor tiempo para su implementación en el robot. Una vez tiene la capacidad de navegar por sí mismo, el siguiente gran paso es otorgarle la habilidad de sortear obstáculos imprevistos. En el mundo real, y teniendo en mente la aplicación de estos robots en almacenes o invernaderos, estos podrán ser tanto estáticos como dinámicos. Aunando estos dos requerimientos, se ha demostrado que los algoritmos de navegación basados en bandas elásticas temporales ofrecen una gran velocidad y robustez en la navegación autónoma a la vez

que muestran gran habilidad en el sorteo de obstáculos. Esto permite una navegación multi-agente en un mismo espacio físico, ya que se detectan mutuamente como impedimentos en su trayectoria y la modifican para no colisionar. Este control se ha implementado a nivel local, para asegurar una mayor tolerancia a fallos en caso de problemas de comunicación con el servidor central.

Finalmente, la cooperación de la que se hablaba al inicio de este resumen se alcanza mediante un reparto eficiente de las tareas a realizar. El enfoque adoptado está basado en un esquema de subasta y licitador, en el que cada tarea es ejecutada por aquel robot que muestre una mayor predisposición para ello. De esta forma, se logra reducir a la mitad el tiempo requerido para llevar a cabo una serie de tareas de transporte cuando se duplica el número de robots en el sistema. Estos resultados se han comprobado tanto en simulación como con los robots físicos, validando así la implementación del sistema multi-robot.

Abstract

Robotics is the branch of engineering devoted to design, construction and control of machines that can resolve human problems. It is an interdisciplinary research field which, combining various sciences, engineerings and technologies, aims to mimic or extend humans' capacities.

There are countless chores in which robots show a better performance than human beings, as the industrial sector is well aware of. Their speed, precision, and repeatability in monotonous jobs are in a superior order of magnitude. In contrast, their hability to establish links and coordinate between themselves and with their environment is still a pending task.

In this regard, this work reveals the diverse and interrelated problems that arise when trying to merge cooperatively the efforts of a multi-agent system towards a common goal. These challenges begin with the perception of the world and the way to model it. With lidar-based simultaneous localization and mapping (SLAM), as for this project, a faithful description of the robots' surroundings is accomplished. However, this devices present some limitations under certain conditions.

Building a map is the essential pillar in which an efficient autonomous navigation bases its foundations. Navigation algorithms have been studied since the dawn of mobile robotics, but each of them offers a different approach that fits a particular requirement. For that reason, several experiments were carried out to determine the most suitable one in terms of time spent for its later implementation in the real robots. Once this demand is met, the next big step is to give the robot the hability to avoid unforeseen obstacles. In the real world, and minding the application of the robots in warehouses or greenhouses environments, these can be static or dynamic. Joining those two specifications, it has been shown that time elastic bands navigation algorithms are a great solution in means of speed, robustness in autonomous navigation as well as in the capability of obstacles avoidance. This allows multi-agent navigation in the same confined space, as they can detect each other and modify their

trajectory to steer clear of the other’s. This control has been developed in a local way for a safer fault tolerance in case of communication issues with the central server.

Finally, the cooperation this abstract started with, is achieved with an efficient distribution of tasks. This approach is based in a auctioneer and bidder scheme, in which each chore is executed by the most suited agent of the system. Therefore, the time required to perform these transport jobs is halved when the number of robots doubles. These results have been tested in simulation and real experiments, which validates the implementation of the multi-robot system.

"*No one can whistle a symphony. It takes a whole orquestra to play it.*"

— H. E. Luccock

Capítulo 1

Introducción

1.1. Interés y motivación del Trabajo Fin de Grado

El sector industrial es uno de los más competitivos a escala mundial, con un mercado en constante cambio, y en el que triunfan aquellos que más rápido se adaptan y mejor cubren las necesidades de los consumidores y las empresas. Por ello, es también uno de los que mayor avidez ha mostrado históricamente por incorporar los últimos avances tecnológicos a su actividad.

Lejos quedan las primeras máquinas de vapor que sustituyeron a los animales a finales del siglo XVIII, o los comienzos de la producción en masa con las cadenas de montaje del Ford T en 1908. En la actualidad, la implementación de nuevas tecnologías en la industria es prácticamente indispensable en un contexto en el que la optimización de recursos es vital para ofrecer productos y bienes con garantías de éxito.

En este ambiente, juegan un gran papel el transporte y la logística fuera de las fábricas, pero también dentro de ellas. A la hora de agilizar el uso de material, la robótica tiene un rol principal tanto en su manipulación, con el uso cada vez más extendido de brazos robóticos en diversas aplicaciones como en su distribución, mediante robots móviles.

Uno de los usos más frecuentes de esta robótica móvil aplicada a la logística es en la gestión y distribución de material en almacenes. En líneas generales, se hace extensible a entornos en los que la materia a transportar tiene un peso considerable para ser ejecutada por un operador humano o las exigencias de volumen de trabajo son simplemente inasumibles para un plantilla de trabajadores. Debido al alto coste de la robótica, esta alternativa es rentable en operaciones de gran escala, en las que a su vez el beneficio aumenta cuanto mayor sea el número de agentes que puedan

trabajar simultáneamente, en lo que se conoce como sistemas multi-robot. Uno de los más populares es el que forman en los almacenes de Amazon los robots Kiva (Galisteo, 2016). Estos sistemas permiten minimizar drásticamente el tiempo invertido en estas tareas y constituyen una gran ventaja competitiva.

Por ello, la motivación del presente Trabajo Fin de Grado es el diseño, simulación e implementación de un sistema multi-robot orientado a tareas de logística. Esta implementación será desde un punto de vista de prototipo mediante los robots móviles iRobot® Create 2.

1.2. Objetivos

El principal objetivo del presente trabajo es implementar un sistema multi-agente real constituido por robots autónomos basados en los robots móviles iRobot® Create 2. Se estudiarán varias alternativas y se analizarán los resultados para determinar el mejor esquema de control que permita una navegación fluida, evitando obstáculos tanto estáticos como dinámicos. Para alcanzarlo, se plantean una serie de subobjetivos:

1. Estudio bibliográfico.

   Se realizará un estudio del estado del arte revisando trabajos previos relacionados con el mapeado, localización, navegación y comportamiento cooperativo de flotas de robots.

2. Estudio del entorno de trabajo ROS (Robot Operating System).

   Para realizar una implementación sólida de este sistema, se ha elegido el framework ROS debido a su robustez, gestión de la comunicación entre los nodos, variedad de bibliotecas disponibles y apoyo entre su comunidad en la red.

3. Estudio del robot iRobot Create 2.

   Se revisarán las características del robot, analizando las capacidades y el ámbito de trabajo que permiten sus sensores y actuadores integrados.

4. Diseño y modelado del prototipo.

   Se diseñará la estructura de dispositivos periféricos (lidar, raspberry, baterías, ...) que permitan llevar a cabo el trabajo. Una vez se instalen todos los dispositivos en el robot, se realizará un modelado tanto funcional como visual del prototipo en simulación.

5. Mapeado del entorno real.

   Se construirá un mapa del entorno real para su posterior uso en la navegación autónoma.

6. Simulación de la navegación autónoma y del sistema multi-robot.

   Para realizar un estudio comparativo de los algoritmos de navegación autónoma y planificación cooperativa se simulará en el entorno Gazebo. Se ha elegido este simulador debido a su potencia y a su compatibilidad con ROS.

7. Implementación en los robots reales.

   Una vez se haya corroborado el correcto funcionamiento en simulación, se procederá a implementar estos algoritmos en el robot real. Para ello, se establecerá la configuración necesaria tanto de la red de equipos como del sistema operativo de cada uno de ellos. De igual forma, se programará todo el código necesario para cumplir con las especificaciones funcionales.

8. Análisis de los resultados obtenidos.

   Se analizarán los resultados del trabajo y se validará su funcionamiento real. Asimismo, se propondrán posibles trabajos futuros de mejora.

9. Elaboración de la documentación del proyecto.

   Se confeccionará una memoria detallada con la explicación de los procedimientos seguidos a lo largo del proyecto.

1.3. Contexto

Siguiendo la línea introducida en el apartado 1.1, el presente trabajo asienta sus raíces en un contexto industrial muy marcado por el auge incesante de la robótica. Desde un punto de vista cuantitativo, por el número de robots presentes, como cualitativo, debido a las aplicaciones cada vez más diversas de estos, el interés por adquirir e integrar este tipo de dispositivos no ha hecho más que incrementar en la última década.

Tanto es así, que según el informe ejecutivo de 2019 de la Federación Internacional de la Robótica (IFR), en 2018 se batió un nuevo récord en el número de robots instalados a nivel mundial, llegando a las 422.271 unidades. Esto supone un aumento del 6 %, valorado en 16.500 millones de dólares y estableciendo el stock operacional de robots en 2.439.543 unidades, un 15 % más (International Federation of Robotics [IFR], 2019a). Este resultado es especialmente relevante si se tiene en cuenta la

creciente incertidumbre por el conflicto comercial entre dos de los mayores destinos: Estados Unidos y China. No obstante, la industrial automóvil se sigue imponiendo como el principal cliente industrial con un 30 % de las instalaciones, seguido por el sector eléctrico/electrónico (25 %) y el del metal y maquinaria (10 %), como se aprecia en la figura 1.3.1.

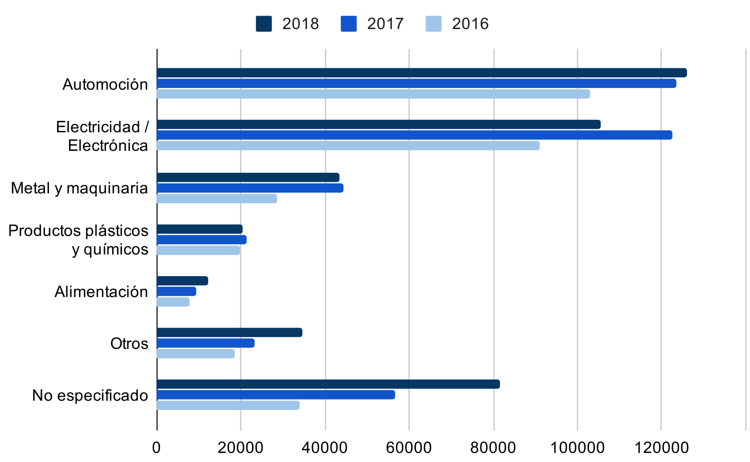


Figura 1.3.1: Instalación anual de robots por industrias (2016 - 2018)

Desde 2010, la demanda de robots industriales ha crecido considerablemente como resultado de la tendencia actual hacia la automatización y las continuas innovaciones técnicas en robots industriales. Desde 2013 hasta 2018, las instalaciones anuales aumentaron en un 19 % de media por año, como se demuestra en la figura 1.3.2 (IFR, 2019a).

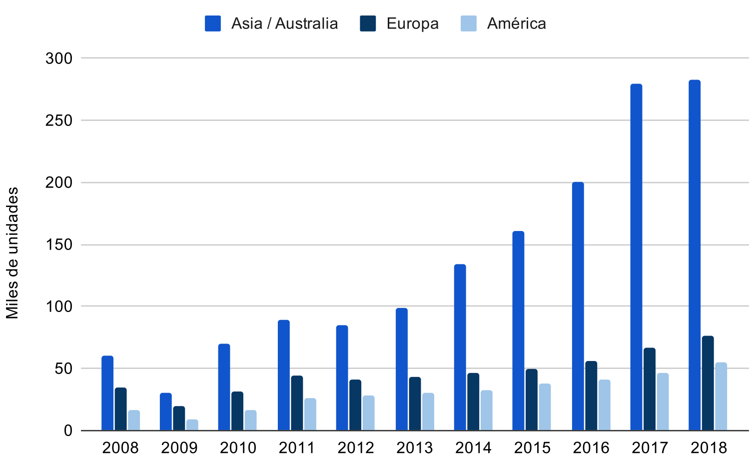


Figura 1.3.2: Instalación anual de robots industriales en el mundo (2016 - 2018)

Centrando de nuevo el foco en el tema principal del trabajo, la robótica móvil es de gran utilidad en un amplio rango de aplicaciones com almacenes y centros de distribución, manufacturación, agricultura y otros entornos (especialmente en logística en hospitales o comercio minorista). En el ámbito de los robots de servicio, entre los que se enmarcan los robots móviles autónomos (AMR), este crecimiento es todavía más palpable. Según se desprende de la conferencia de prensa de la Federación Internacional de la Robótica en 2019 (IFR, 2019b), este sector tuvo un crecimiento del 90 % en unidades vendidas en 2018, proyectando en torno al 40 % en crecimiento anual compuesto para 2022 (ámbito doméstico y profesional), como se refleja en las figuras 1.3.3 y 1.3.4.

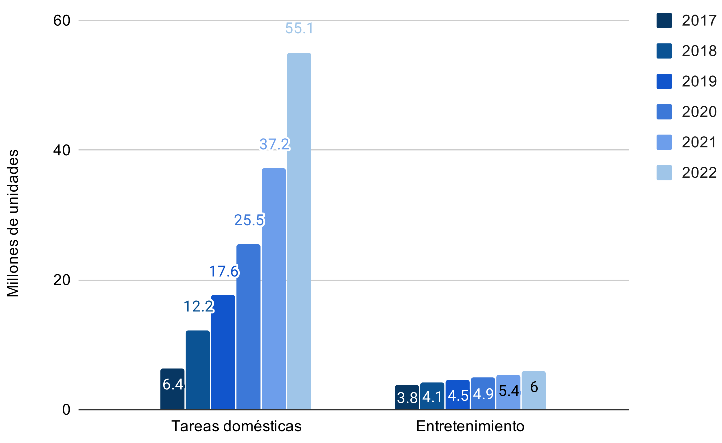


Figura 1.3.3: Robots de servicio vendidas para uso personal / doméstico. Unidades vendidas en 2017 y 2018, previsión para 2019-2022

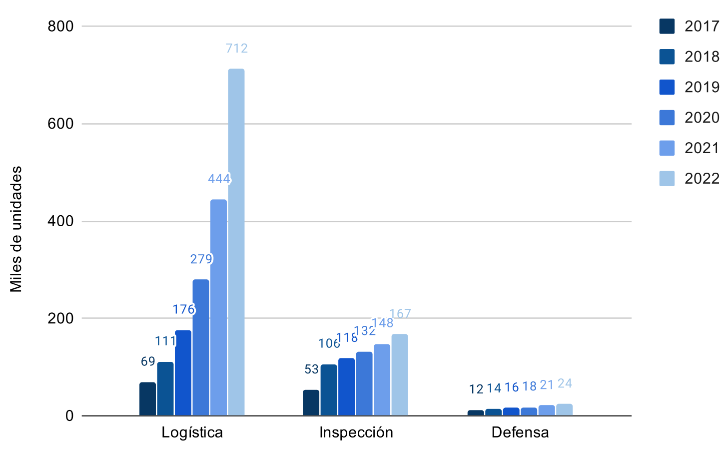


Figura 1.3.4: Robots de servicio vendidas para uso profesional. Unidades vendidas en 2017 y 2018, previsión para 2019-2022

No obstante, de forma paralela a la ejecución de este trabajo se ha desarrollado un acontecimiento a escala mundial y proporciones históricas: la pandemia global causada por el virus SARS-CoV-2. Las consecuencias de esta pandemia pasarían por una crisis económica más que previsible, por lo que las cifras estimadas anteriormente para los años venideros se verán afectadas en alguna medida. Sin embargo, esta situación nunca antes vista ha puesto de manifiesto una nueva razón para apostar por la robótica en ciertos puestos de riesgo: el temor al contagio.

Por esta razón, la velocidad con la que suceda esta sustitución podría incrementarse con motivo de las medidas de distanciamiento social y la imposibilidad de desempeñar ciertos trabajos de gran concentración de personal. A su vez, son cada vez más las soluciones que incorporan la robótica móvil, por ejemplo, en aspectos como la desinfección de espacios públicos (Ackerman, 2020; Thomas, 2020; Yang et al., 2020).

Adicionalmente, este trabajo se enmarca dentro del proyecto AGRICOBIOT (Moreno, Giménez y Rodríguez, 2019), cuyo objetivo es la construcción de un robot colaborativo para el transporte inteligente en interior de invernaderos con soporte en IoT. En este entorno, la planificación de un sistema multi-robot será fundamental para realizar tareas de transporte de productos.

1.4. Resumen de resultados

El resultado fundamental de este trabajo es la puesta en funcionamiento de un sistema multi-robot autónomo y cooperativo. Algunas de las pruebas que corroboran su desempeño, tanto en simulación como reales, se pueden visualizar en (Grupo de investigación ”Automática, Robótica y Mecatrónica” de la UAL [ARM UAL], 2020a, 2020b, 2020c). En ellas se observa cómo el sistema distribuye una serie de posiciones objetivos entre sus dos agentes, sin colisiones de ningún tipo entre ellos ni con obstáculos del entorno, tanto estáticos como dinámicos. En la figura 1.4.1 se muestra el impacto que tiene el número de robots involucrados en el tiempo de ejecución total. Esta disminución es aproximadamente igual a la proporción de agentes del sistema. Algo similar ocurre en la distancia recorrida (figura 1.4.2).

Para respaldar estas demostraciones se ha llevado a cabo una serie de pruebas en simulación. En ellas se ha aumentado el número de robots en el sistema hasta cuatro. De igual modo, los tiempos y distancias se han reducido en cuatro aproximadamente para el caso con más robots, como cabía esperar (figuras 1.4.3 y 1.4.4).

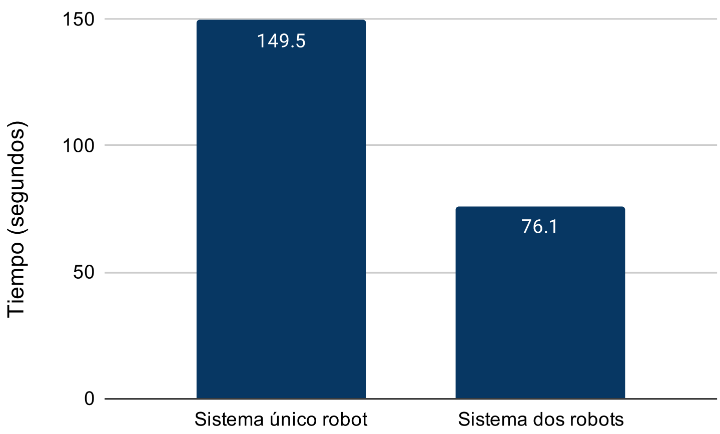


Figura 1.4.1: Comparativa de tiempo invertido por el sistema real

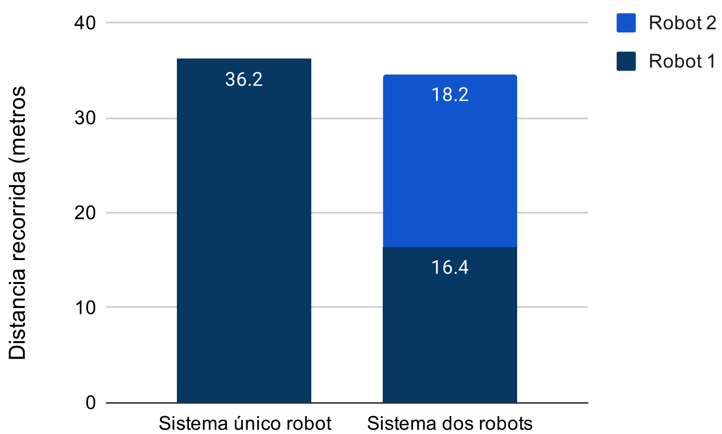


Figura 1.4.2: Comparativa de distancia recorrida por el sistema real

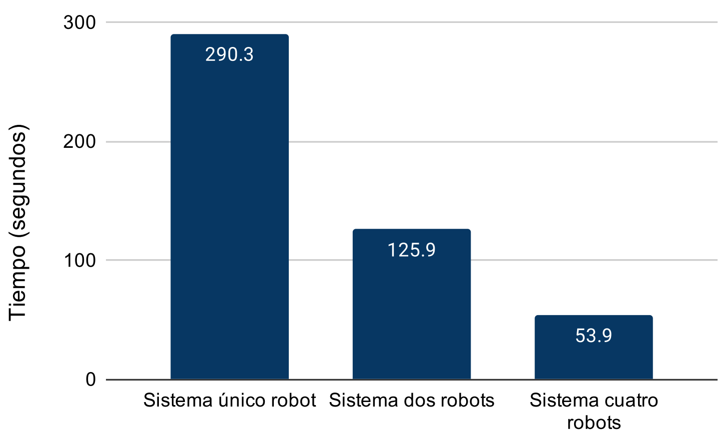


Figura 1.4.3: Comparativa de tiempo invertido por el sistema en simulación

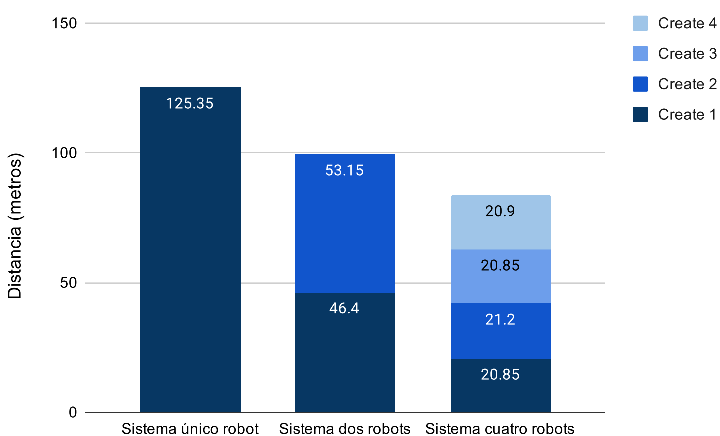


Figura 1.4.4: Comparativa de distancia recorrida por el sistema en simulación

Tras un primer análisis de los resultados obtenidos, se considera probada la efectividad del sistema para cumplir satisfactoriamente los objetivos de un reparto equilibrado y autónomo de tareas. Además, cabe destacar que en todas las pruebas se han ejecutado los desplazamientos sin colisiones de ningún tipo.

1.5. Planificación temporal

En este apartado se presenta un desglose aproximado del tiempo invertido por cada actividad en la realización de este trabajo en la tabla 1.5.1. El cómputo total de horas excede ligeramente del establecido para un Trabajo de Fin de Grado, estipulado en 300 horas en el Grado de Ingeniería Electrónica Industrial.

1.6. Competencias utilizadas en el TFG

A lo largo de las diferentes asignaturas que se cursan en el grado de Ingeniería Electrónica Industrial, el estudiante adquiere una serie de competencias que le permiten el ejercicio de la actividad profesional conforme a las exigencias y estándares empleados en la ingeniería. Estas se agrupan en tres grandes grupos: competencias básicas, transversales y específicas. A continuación se indicarán las competencias que han resultado primordiales para afrontar el Trabajo Fin de Grado.

Las competencias de carácter básica, para todos los títulos de grado, vienen definidas en el R.D. 1393/2007, de 29 de octubre, y están dirigidas a la adquisición por el es-

	Planificación semanal																														
	2019								2020																						
	Noviembre				Diciembre				Enero					Febrero				Marzo				Abril					Mayo				
Actividad	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	
Estudio del estado del arte	5	5	5	5											2	2	2	2	2							5					35
Estudio del entorno de trabajo ROS	5	5	5	5	5	5									5	5	5	5	5	5	5										65
Estudio del robot iRobot Create 2				5	5	5																									15
Diseño y modelado del prototipo					5	5	5																								15
Mapeado virtual del entorno						10	5																								15
Configuración de la red de equipos							10								5																15
Mapeado real de entorno								10																							10
Simulación de la navegación autónoma								5	10						5																20
Implementación real de la navegación autónoma															10	10	5	5	5				5								40
Configuración de algoritmos de navegación autónoma																5	10	10	5	5	5	5	5	5							55
Simulación de la planificación multi-robot																									5	15	5				25
Implementación real de la planificación multi-robot																											10	5			15
Grabación de pruebas reales																											3	2			5
Análisis de los resultados obtenidos																											5	5			10
Elaboración de la documentación del proyecto																						5	5	5	5	5	10	20	5		60
Resolución de problemas y errores																		5	5	5	4						1				20
	10	10	10	15	15	25	20	15	10	0	0	0	0	0	27	22	22	27	22	15	14	10	15	10	10	25	34	32	5	0	
	45				75				10					71				78				70					71				420

Tabla 1.5.1: Cronograma temporal

tudiante de una formación general, en una o varias disciplinas, orientada a la preparación para el ejercicio de actividades de carácter profesional.

- CB1. Poseer y comprender conocimientos.
- CB2. Aplicación de conocimientos.
- CB3. Capacidad de emitir juicios.
- CB4. Capacidad de comunicar y aptitud social.
- CB5. Habilidad para el aprendizaje.

Las competencias transversales de la Universidad de Almería.

- UAL1. Conocimientos básicos de la profesión.
- UAL2. Habilidad en el uso de las TIC.
- UAL3. Capacidad para resolver problemas.
- UAL4. Comunicación oral y escrita en la propia lengua.
- UAL5. Capacidad de crítica y autocrítica.
- UAL6. Trabajo en equipo.
- UAL7. Conocimiento de una segunda lengua.
- UAL8. Compromiso ético.
- UAL9. Capacidad para aprender a trabajar de forma autónoma
- UAL10. Competencia social y ciudadanía global.

En cuanto a las competencias específicas del grado, las relativas a este trabajo son:

- CT3. Conocimiento en materias básicas y tecnológicas, que les capacite para el aprendizaje de nuevos métodos y teorías, y les dote de versatilidad para adaptarse a nuevas situaciones.
- CT4. Capacidad de resolver problemas con iniciativa, toma de decisiones, creatividad, razonamiento crítico y de comunicar y transmitir conocimientos, habilidades y destrezas en el campo de la Ingeniería Industrial.

- CT7. Capacidad de analizar y valorar el impacto social y medioambiental de las soluciones técnicas.
- CT10. Capacidad de trabajar en un entorno multilingüe y multidisciplinar.
- CB1. Capacidad para la resolución de los problemas matemáticos que puedan plantearse en la ingeniería. Aptitud para aplicar los conocimientos sobre: álgebra lineal; geometría; geometría diferencial; cálculo diferencial e integral; ecuaciones diferenciales y en derivadas parciales; métodos numéricos; algorítmica numérica; estadística y optimización.
- CB3. Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación en ingeniería.
- CB5. Capacidad de visión espacial y conocimiento de las técnicas de representación gráfica, tanto por métodos tradicionales de geometría métrica y geometría descriptiva, como mediante las aplicaciones de diseño asistido por ordenador.
- CRI4. Conocimiento y utilización de los principios de teoría de circuitos y máquinas eléctricas.
- CRI5. Conocimientos de los fundamentos de la electrónica.
- CRI6. Conocimientos sobre los fundamentos de automatismos y métodos de control.
- CRI7. Conocimiento de los principios de teoría de máquinas y mecanismos.
- CTEE3. Conocimiento de los fundamentos y aplicaciones de la electrónica digital y microprocesadores.
- CTEE4. Conocimiento aplicado de electrónica de potencia.
- CTEE5. Conocimiento aplicado de instrumentación electrónica.
- CTEE7. Conocimiento y capacidad para el modelado y simulación de sistemas.
- CTEE8. Conocimientos de regulación automática y técnicas de control y su aplicación a la automatización industrial.
- CTEE9. Conocimientos de principios y aplicaciones de los sistemas robotizados.
- CTEE10. Conocimiento aplicado de informática industrial y comunicaciones.
- CTEE11. Capacidad para diseñar sistemas de control y automatización industrial.

- TFG. Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería Industrial de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.

1.7. Estructura del Trabajo Fin de Grado

El presente trabajo técnico consta de cinco capítulos, cuyo contenido se detalla a continuación:

- En el primer capítulo se expone la motivación, interés y resultados de este trabajo, así como un breve resumen de los resultados alcanzados.
- En el segundo capítulo se lleva a cabo una completa revisión del estado del arte de la robótica cooperativa y de todos los mecanismos y algoritmos necesarios para resolver cada uno de los subproblemas en los que se divide este proyecto.
- En tercer capítulo se indican los dispositivos y métodos empleados en el diseño e implementación del sistema, incluyendo los paquetes ROS y bibliotecas.
- En el cuarto capítulo se explica exhaustivamente cómo se han trasladado a código estos métodos, se analizan los resultados derivados de este trabajo y se valida el diseño propuesto.
- El quinto y último capítulo recoge las conclusiones y futuras líneas de trabajo que emanan de este proyecto.

Capítulo 2

Estado del arte

Como se describirá con más detalle en el capítulo 4, la implementación de un sistema multi-robot es un problema multidisciplinar que abarca una gran variedad de subproblemas que es necesario abordar. De forma resumida, son los siguientes:

- Mapeado del entorno (SLAM)
- Localización del robot en el mapa
- Planificación de rutas y navegación autónoma
- Planificación multi-robot y asignación de tareas

2.1. Simultaneous Localization And Mapping (SLAM)

El SLAM (acrónimo de *Simultaneous Localization and Mapping*) comprende la estimación simultánea del estado de un robot equipado con sensores integrados, y la construcción de un modelo (mapa) del entorno que los sensores están percibiendo (Cadena et al., 2016). En casos simples, el estado del robot es descrito por su pose (posición y orientación), aunque se puede incluir una mayor información en dicho estado, como la velocidad del robot, errores o *biases* del sensor y parámetros de calibración. El mapa, por otro lado, es una representación de aspectos de interés, como la posición de puntos de referencia u obstáculos, que describe el entorno en el que opera el robot.

La necesidad de usar un mapa del entorno es doble. Primeramente, se requiere el mapa para realizar otras tareas. Por ejemplo, un mapa puede proveer la información necesaria al planificador de trayectorias o mostrar de forma intuitiva la presencia de

un operador humano. En segundo lugar, el mapa permite limitar el error cometido al estimar el estado de el robot. En ausencia de un mapa, la navegación basada exclusivamente en la odometría deriva rápidamente con el tiempo, mientras que usando un mapa el robot puede "restablecer" su error de localización al volver a visitar áreas conocidas (cierre de bucle). Por lo tanto, las aplicaciones del SLAM se encuentran en todos los escenarios en los que un mapa *a priori* no está disponible y necesita ser construido.

Respecto al estado del arte, en (Cadena et al., 2016) se realiza un trabajo de grandísimo detalle sobre los inicios, progresos y retos por resolver de las investigaciones más loables sobre el tema. Se repasarán brevemente algunos de ellos.

Durrant-Whyte y Bailey ofrecen una revisión histórica exhaustiva de los primeros 20 años del problema SLAM en dos artículos (Durrant-Whyte y Bailey, 2006a) y (Durrant-Whyte y Bailey, 2006b). Estos cubren principalmente lo que se denomina la edad clásica (1986-2004) en la cual se introdujeron las formulaciones probabilísticas de SLAM. Éstas incluían enfoques basados en filtros de Kalman extendidos y filtros de partículas Rao-Blackwellised y estimaciones de máxima probabilidad.

El siguiente período (2004-2015) es lo que se podría llamar la edad del análisis algorítmico (Dissanayake et al., 2011). En él se estudiaron las propiedades fundamentales del SLAM, incluyendo la convergencia de la observabilidad y la consistencia. El aspecto clave de este período fue el desarrollo de solucionadores de SLAM eficientes y de bibliotecas de código abierto.

La arquitectura básica de un sistema SLAM moderno incluye dos componentes principales: el *front-end* y el *back-end*. El front-end abstrae los datos recogidos por los sensores formando modelos completos sobre los que realizar estimaciones, mientras que el back-end lleva a cabo inferencias sobre estos datos producidos por el front-end.

La estándar *de facto* de los sistemas SLAM en la actualidad basa su formulación en un problema de estimación máxima *a posteriori*, MAP, (Gutmann y Konolige, 1999; F. Lu y Milios, 1997) usando a menudo el formalismo de *factor graph* para relacionar la interdependencia de las variables. Supóngase que se pretende estimar una variable X que representa la trayectoria del robot, como un conjunto de poses, y la posición de ciertos puntos de referencia en el entorno. Dadas una serie de mediciones $Z = \{z_k : k = 1, ..., m\}$ tal que cada medición puede expresarse como función de X, $z_k = h_k(X_k) + \epsilon_k$, donde $X_k \subseteq X$ es un subconjunto de las variables, h_k es una función conocida y ϵ_k es una medida de ruido aleatoria.

En la estimación MAP, se estima la variable X calculando la asignación de variables X^* que consigue una posterior probabilidad máxima $p(X|Z)$ (ecuación 2.1.1):

$$X^* = \operatorname*{argmax}_x p(X|Z) = \operatorname*{argmax}_X p(Z|X)p(X) \tag{2.1.1}$$

donde la igualdad sigue el teorema de Bayes. $p(Z|X)$ es la probabilidad de las medidas Z dadas las asignaciones X, y $p(X)$ es la probabilidad anterior sobre X. Esta probabilidad incluye todo el conocimiento previo sobre X, y si no lo hubiera, se convertiría en una constante (distribución uniforme). Al contrario de lo que ocurre con el filtro de Kalman, la estimación MAP no requiere una distinción explícita entre los modelos de movimiento y observación. Ambos son tratados como factores e incluidos en el proceso de estimación. Asumiendo que las mediciones Z son independientes y que los ruidos no están correlacionados, la ecuación 2.1.1 se factoriza en:

$$X^* = \operatorname*{argmax}_X p(X) \prod_{k=1}^{m} p(z_k|X) = \operatorname*{argmax}_X p(X) \prod_{k=1}^{m} p(z_k|X_k) \tag{2.1.2}$$

donde z_k solo depende del subconjunto X_k.

La ecuación 2.1.2 puede ser interpretada en términos de inferencia sobre unos *factor graphs* (Kschischang, Frey y Loeliger, 2001). Las variables corresponden a los nodos y los términos $p(z_k|X_k)$ y $p(X)$ son los factores. Un *factor graph* es un modelo gráfico que codifica la dependencia entre el factor k-ésimo (con su correspondiente medida z_k) y las variable X_k.

La formulación del SLAM como un *factor graph* (Cadena et al., 2016) se expresa visualmente en la figura 2.1.1. Los círculos azules representan las poses de robot en los instantes de tiempo $(x_1, x_2, ...)$, los círculos verdes denotan las posiciones de los puntos de referencia $(l_1, l_2, ...)$ y el círculo rojo, la variable asociada con los parámetros de calibración. Los factores se representan como cuadrados negros: la etiqueta ”u” identifica a los factores de las restricciones de la odometría, ”v” para las observaciones de la cámara (o lidar, en el caso del presente trabajo), ”c” para los cierres de bucle y ”p” para los factores anteriores.

Reescribiendo la ecuación 2.1.2 y considerando ϵ_k como un ruido de distribución gaussiana de media cero con una matriz de información Ω_k (inversa de la matriz de covarianza), la probabilidad de la medición resultaria:

$$p(z_k|X_k) \propto \exp(-\frac{1}{2}\|h_k(X_k) - z_k\|^2_{\Omega_k}) \tag{2.1.3}$$

De forma similar, la ecuación 2.1.3 puede reescribirse como

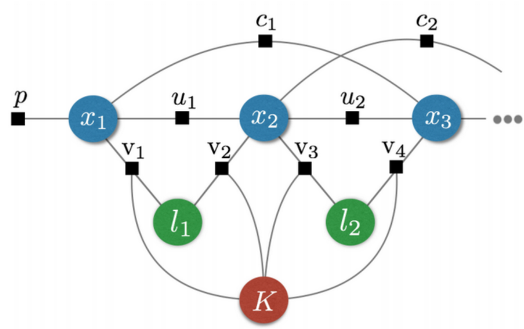


Figura 2.1.1: SLAM como un *factor graph*. (Fuente: Cadena et al., 2016)

$$p(X) \propto \exp\left(-\frac{1}{2}\|h_0(X) - z_0\|^2_{\Omega_0}\right) \tag{2.1.4}$$

para una función h_0 y matriz de información Ω_0. Dado que la maximización es equivalente a la minimización del logaritmo negativo, la estimación MAP se convierte en:

$$X^* = \underset{X}{\operatorname{argmin}} \; -\log\left(p(X)\prod_{k=1}^{m} p(z_k|X_k)\right) = \underset{X}{\operatorname{argmin}} \sum_{k=0}^{m} \|h_k(X_k) - z_k\|^2_{\Omega_k} \tag{2.1.5}$$

lo cual es un problema de mínimos cuadrados no lineal.

En este punto cabe destacar la formulación HMT-SLAM (Blanco-Claraco, Fernández-Madrigal y Gonzalez, 2008) para mapeo a gran escala. La introducción del concepto de recorrido metrico-topológico híbrido (HMT) comprende la secuencia de áreas que ha recorrido el robot (parte topológica) y sus poses en cada una de ellas (parte métrica). Posteriormente, considerando la posterior distribución de todo el recorrido HMT se puede obtener la distribución de probabilidad sobre todas las estructuras topológicas potenciales del entorno. Al igual que los trabajos anteriores, se trata de un enfoque bayesiano en el que los nodos denotan submapas métricos y los arcos (conexión entre nodos) son las transformaciones de coordenadas entre ellos.

Por último, cada vez se introducen nuevas herramientas para sistemas SLAM como resultado de su análisis teórico. Esto da lugar a enfoques basados en SDP (Semidefinite Optimization Problem) (Carlone et al., 2016) o en la relajación convexa para evitar mínimos locales (Rosen, DuHadway y Leonard, 2015).

Con la irrupción de nuevos sensores cada vez más asequibles y precisos, el problema del mapeo y localización simultáneos se ha adaptado para integrarlos y combinar la información recogida por cada uno de ellos. Así, surgen nuevas

formulaciones para el SLAM con cámaras basadas en eventos (Kim, Leutenegger y Davison, 2016). Al incorporar más de un sensor para esta tarea, aparecen nuevos enfoques basados en una estructura modular que permite el empleo de lidar, cámaras monoculares y estéreo, odometría, IMU y GPS (Blanco-Claraco, 2019).

De igual forma, los grandes avances en *Machine Learning* y *Deep Learning* aportan una visión novedosa a este problema. Si bien no significa reemplazar los algoritmos actuales bien conocidos, se ha demostrado la capacidad de estas nuevas herramientas para relacionar los marcos de referencia entre imágenes (Costante et al., 2016) y la localización de los 6 grados de libertad de una cámara mediante *regression forest* (Valentin et al., 2015) y redes neuronales convolucionales (Kendall, Grimes y Cipolla, 2015). Una solución completa sería el popular algoritmo FastSLAM basado en redes neuronales (Li, Song y Hou, 2015).

2.2. Localización del robot

La mayoría de aplicaciones de la robótica móvil requieren un conocimiento sobre su posición en el entorno. Esta localización suele expresarse en relación a un mapa y constituye la base para otros módulos software del robot, los cuales realizan otras tareas de más alto nivel a partir de dicha información. Éste es el punto de partida de un robot plenamente autónomo.

Para ello, es imprescindible un modelo de la automoción del robot, para el cual los diversos sensores que lleve integrados proveerán la información recogida del ambiente. Estos sensores pueden ser propioceptivos, los cuales realizan mediciones sobre el estado propio del robot (encoders, IMU,...) o exteroceptivos, que perciben información del ambiente (sónares, sensores infrarrojos, cámaras, lidar o GPS). En escenarios de interior como el que nos ocupa en el trabajo, el GPS es descartado por la inaccesibilidad de su señal y su amplio margen de error. A su vez, los propios sensores pueden dar lugar a errores, diferenciando los debidos a fuentes deterministas, como una mala parametrización de las dimensiones del robot o estocásticas. Estas últimas, se dan en todo robot móvil que se precie, por fenómenos tales como inclinaciones o irregularidades del terreno, deslizamientos o movimientos externos. Al tratarse de errores no sistemáticos, no es posible una compensación directa, pero pueden ser modelados con cierta incertidumbre estocástica.

En resumen, un robot autónomo comienza desde una posición inicial sin conocimiento previo (o una limitada estimación inicial) del entorno e intenta recabar información a través de las mediciones de sus sensores. Con ello crea una hipótesis

sobre su siguiente estado mediante métodos probabilísticos (en la mayoria de los algoritmos actuales), y es actualizado siguiendo una formulación bayesiana clásica.

A continuación, se comentan algunos de los principales métodos presentados en (Malagon-Soldara et al., 2015). En dicho trabajo se repasan algunos conceptos esenciales de los algoritmos de localización en robots móviles.

Para controlar un robot móvil en ocasiones es necesario combinar información de diversas fuentes, siendo las características y prestaciones de cada una de ellas muy variadas. Para mitigar este problema, se recurre al filtro de Kalman, muy usado en robótica (Melo y Junior, 2010; Roumeliotis y Bekey, 2000).

El filtro de Kalman es un procesamiento de datos de forma recursiva que estima el estado de un sistema lineal dinámico con ruido (Negenborn, 2003). Un ejemplo de ello es la localización de un robot, dadas por sus coordenadas en el mapa y por su orientación. El hecho de que las variable de estado puedan tener ruido y no sean directamente observables complica esta estimación. Para llevarla a cabo, el filtro de Kalman tiene acceso a las mediciones realizadas por los sensores, que están linealmente relacionadas con el estado y distorsionadas por el ruido. Si este ruido es gaussiano, el filtro de Kalman es un estimador estadísticamente óptimo respecto a cualquier medida razonable.

A grandes rasgos, el filtro actúa como una suma ponderada de las mediciones, otorgando mayor confianza a ciertas medidas que a otras en función del sensor o de la situación. En el caso de sistemas no lineales, se emplea el filtro de Kalman extendido (EKF) (Kiriy y Buehler, 2002), el cual involucra la linealización del sistema e incluso de la medida. Otro enfoque es el filtro de Kalman ”sigma punto” (SPKF) para minimizar los posibles errores e inconsistencias (Paul y Wan, 2008).

Otra visión de los algoritmos de localización es la que introducen los filtros de partículas. Los filtros de particulas son métodos secuenciales de Monte Carlo bajo un marco de estimación bayesiana que ha sido utilizado ampliamente en procesamiento de señales y de imagen, localización de robots e incluso en algunas aplicaciones de economía.

El metodo de Monte Carlo fue propuesto en el año 1949 y se basa en la aproximación de expresiones matemáticas complejas mediante métodos estadísticos. *”La idea de emplear un enfoque estadístico [...] es referida como el método de Monte Carlo. [...] El proceso computacional es como sigue: imagínese que se tiene un grupo de partículas representadas cada una con un número. Cada uno de estos números denota el tiempo, los vectores de posición y velocidad, así como un índice de la naturaleza de cada partícula. Con cada uno de estos conjuntos de números se inicia un proceso aleatorio que conducen a la*

determinación de un nuevo conjunto de valores. Existe, de hecho un grupo de distribuciones para los nuevos valores de los parámetros trascurrido un intervalo de tiempo Δt. Imagine que se extraen, de forma aleatoria e independiente, valores de una colección preparada según esas distribuciones. [..] Considerando un número grande de partículas [...] se espera obtener un buena muestra de las distribuciones en el tiempo $t + \Delta t$. [...] La característica principal del proceso es evitar tener que lidiar con múltiples integraciones y multiplicaciones, y en su lugar extraer simples cadenas de eventos." (Metropolis y Ulam, 1949)

En esencia, la idea clave es representar la siguiente función de densidad de probabilidad de las variables de estado con un conjunto de partículas con pesos asociados y realizar una estimación basada en ellas. Los modelos calculados de esta forma son difícilmente reproducibles con filtros de Kalman debido a las no linealidades y aleatoriedades.

El algoritmo de localización de Monte Carlo (MCL) quedaría, expresado en pseudocódigo, tal y como se muestra en la tabla 2.2.1 (Thrun et al., 2001):

$X' = \Theta$
for $i = 0$ to m do
  generar aleatoriamente x a partir de X según $w1, w2, ...$
  generar aleatoriamente $x' \sim p(x'|a, x)$
  $w' = p(o|x')$
  añadir $\langle x', w' \rangle$ a X'
endfor
normalizar los factores de importancia (suma=1) en X'
return X'

Tabla 2.2.1: Pseudocódigo del algoritmo MCL(X, a, o)

siendo x y X el estado y su distribución de probabilidad; x' y X', el estado esperado y su densidad de probabilidad; a, las acciones; o, las observaciones y w, los factores de importancia o pesos.

Una adaptación de este algoritmo es el método de Monte Carlo adaptativo (AMCL) o de muestreo KLD (Pfaff, Burgard y Fox, 2006). En este trabajo se introduce un modelo adaptativo de los sensores que tiene en cuenta explícitamente la incertidumbre de la localización debido a la representación basada en muestras. Para calcular esta incertidumbre se emplean técnicas basadas en estimación de densidades. La mayoría de alternativas estiman la densidad en un punto x considerando una región a su alrededor en la que el tamaño de dicha región depende

del número y de la densidad local de las muestras. Esta estimación de densidades es una forma de calcular una incertidumbre adicional debido a este tipo de representación. Como resultado, la función de probabilidad se adapta automáticamente a la densidad local de muestras, siendo suave en la localización global y rápida en el seguimiento de posición.

2.3. Navegación autónoma

En la ingeniería, y más concretamente en la robótica, el concepto de autonomía de los robots móviles abarca muchas áreas de conocimiento, metodologías y, en última instancia, algoritmos diseñados para el control de trayectoria, el sorteo de obstáculos, la localización, la construcción de mapas, etc. En una gran medida, el éxito de una misión de planificación de ruta y navegación de un robot móvil autónomo depende de la disponibilidad de una representación precisa del entorno de navegación (Mohanty y Parhi, 2013).

La consecución de esta autonomía en la navegación de un robot radica en un concepto esencial: la planificación.

2.3.1. Planificación de la navegación autónoma

Los términos planificación de movimiento y planificación de trayectoria se utilizan a menudo para este tipo de problemas. En ocasiones se hace referencia a una versión clásica de la planificación del movimiento como el problema del movedor del piano, el cual se incluye como introducción ilustrativa (LaValle, 2006). Supóngase que se da un modelo CAD (Computer Aided Design) preciso de una casa y un piano como entradas a un algoritmo. Este algoritmo debe determinar cómo mover el piano de una habitación a otra en la casa sin golpear ningún objeto. La planificación del movimiento del robot ignora generalmente la dinámica y otras restricciones diferenciales y se centra principalmente en las traslaciones y rotaciones requeridas para mover el piano. En trabajos recientes, sin embargo, se incluyen otros aspectos, tales como incertidumbres, restricciones diferenciales, errores de modelado y diversas optimizaciones. La planificación de trayectoria, por otro lado, se refiere al problema de tomar la solución desde un algoritmo de planificación de movimiento del robot y determinar cómo moverse a lo largo de dicha solución de manera que se respeten las limitaciones mecánicas del robot.

A lo largo del libro (LaValle, 2006) se ofrece una amplísima revisión de todos los aspectos concernientes a la planificación y navegación autónoma. Siguiendo algunos de sus párrafos, se incluyen algunos de los conceptos centrales de esta planificación.

- Estado: es la representación de todas las posibles situaciones en las que pueda encontrarse el robot. Generalmente esta constituido por su posición, orientación y velocidad. En la mayoría de aplicaciones este espacio de estados está implícito en el algoritmo de planificación.
- Tiempo: todos los problemas de planificación requieren una secuencia de decisiones que deben ser tomadas a lo largo del tiempo. En ocasiones, el tiempo es una varible explícita del problema, como puede ser la conducción más rápida posible entre dos puntos o simplemente como reflejo de una serie de acciones que deben realizarse de forma secuencial.
- Acciones: los planes de movimiento o trayectoria generan acciones a realizar por parte del robot, lo cual altera su estado. Éstas pueden ser modeladas como una conjunto de acciones discretas o siguiendo ciertas ecuaciones diferenciales a lo largo del tiempo.
- Estados iniciales y finales: por norma general, los problemas de planificación parten de un punto inicial con el objetivo de llegar a un punto o estado final.
- Criterio: es una forma de codificar el resultado esperado. Suelen reducirse a dos, principalmente. El primero de ellos es la factibilidad, en el cual se busca encontrar el camino que lleve al robot desde el estado inicial hasta el final, independientemente de su eficiencia (Likhachev y Ferguson, 2009). La otra alternativa es la óptima, en la cual se trata de optimizar cierto rendimiento (Jan, Chang y Parberry, 2008).
- Plan: es la estrategia que debe seguir el robot. Suele ser un conjunto de poses a lo largo de un camino o una secuencia de acciones.
- Planificador: es la herramienta encargada de las construcción del plan.
- Arquitectura: la estructura de control puede plantearse desde un punto de vista reactivo o local, cuando el planificador responde únicamente frente a la información que le llega de los sensores y sin un modelo del entorno; deliberativo o global, cuando sí existe este modelo y la planificación se realiza por completo antes de que el robot comience a moverse, e híbrida, cuando se combinan planificadores globales y locales, generalmente con un mayor rendimiento.

- Algoritmo de planificación: es el corazón de la planificación. Tiene en cuenta todos los aspectos del robot y del entorno en el que se mueve para producir una solución efectiva al problema de la navegación autónoma. Existe una infinidad de algoritmos propuestos a lo largo de toda la literatura, entre los que se distinguen los métodos clásicos o heurísticos (evolutivos).

Estos dos últimos puntos son los que se van a desarrollar a continuación, tratando de cubrir los principales enfoques.

2.3.2. Arquitecturas de control

En lo que a arquitectura de control se refiere, se puede hablar de una navegación reactiva cuando se generan acciones de control basadas en la información actual de los sensores. Para ello, se confecciona un modelo local del ambiente sin un proceso de planificación en niveles superiores que vaya guiando desde una perspectiva completa. Se evita así la construcción de un modelo global y se simplifican los cálculos, ya que el robot se limita a reaccionar, como su propio nombre indica, a los fenómenos que se encuentre en su camino (figura 2.3.1). Fue introducida por primera vez hace más de 30 años (Brooks, 1986) en el Massachusetts Institute of Technology (MIT). Este paradigma es el más simple, y por ello no siempre converge hacia el objetivo final, pudiendo detenerse al alcanzar mínimos locales. Algunos de las perspectivas más clásicas se introducen en (Balch, 1993) o basadas en métodos heurísticos de lógica difusa como (Song y Sheen, 2000).

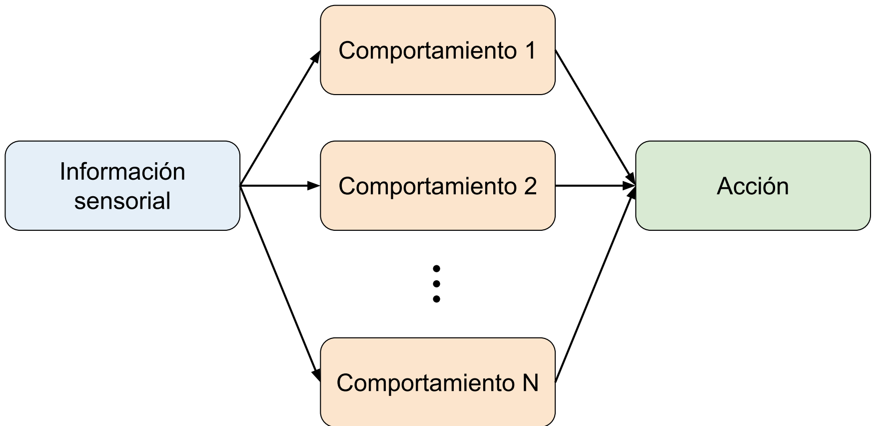


Figura 2.3.1: Esquema de navegación reactiva

Por el contrario, en una navegación deliberativa o de planificación global se emplea un modelo del mundo en el que se mueve el robot junto con información (o no) de los sensores a bordo.

La arquitectura de control deliberativo está compuesta por tres módulos: los módulos de detección, planificación y acción (figura 2.3.2). Primero, el robot detecta su entorno y crea un modelo global y estático combinando la información sensorial. Posteriormente, se ejecuta el módulo de planificación para buscar un camino óptimo hacia el siguiente objetivo y se genera un plan para el robot. Finalmente, el robot ejecuta las acciones calculadas para alcanzar el objetivo. Si el resultado es el esperado, el robot se detiene y actualiza la información para realizar el siguiente movimiento. De esta forma, repite el proceso hasta que alcanza la meta (Nakhaeinia et al., 2011). Por ejemplo, (Lindemann y LaValle, 2005a) introduce una visión centrada en una combinación de campos de vectores o (Lindemann y LaValle, 2005b) basada en robótica en la nube (*cloud robotics*).

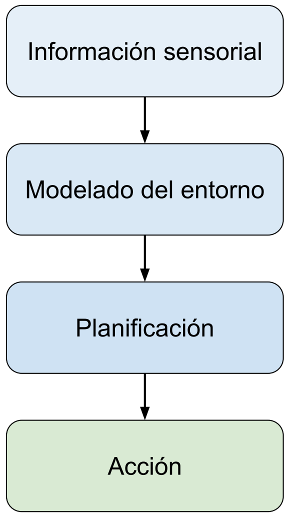


Figura 2.3.2: Esquema de navegación deliberativa

En la práctica, se suele optar por un esquema de control híbrido, que integra un control local que permita una rápida reacción ante aspectos desconocidos del mapa (obstáculos, incertidumbres) y una planificación global que asegure la convergencia hasta el destino.

Las arquitecturas de control híbridas comunes se componen de tres capas: capa deliberativa o global, capa de ejecución de control y capa reactiva o local (figura 2.3.3). La capa deliberativa se aplica en cuestiones de alto nivel para desarrollar un plan óptimo. Las restricciones de alto nivel consisten en la fusión de sensores, la construcción de mapas y la planificación. Luego, los comandos óptimos del nivel superior se envían a la capa reactiva para generar la acción del robot. La capa de ejecución (coordinador de comportamiento) es responsable de supervisar la interacción entre la capa de alto nivel y la capa de bajo nivel. La integración de diversas

características en la arquitectura híbrida forma una arquitectura flexible y robusta para los sistemas de control (Nakhaeinia et al., 2011). Algunas estrategias más tradicionales se pueden ver en (Connell et al., 1992) mediante la llamada arquitectura SSS (*servo, subumption, symbolic*), así como otras más modernas, basadas en conceptos como el *Q-Learning* (uno de los pilares iniciales del *Reinforcement Learning*), en este caso de tipo jerárquico (HQL) en (Chen, Li y Dong, 2008).

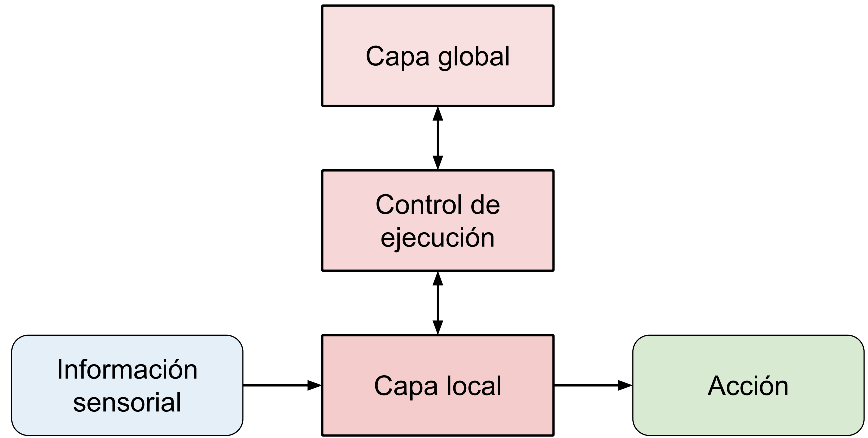


Figura 2.3.3: Esquema de navegación híbrida

En el caso de un esquema de control deliberativo con un modelo del entorno, el procesamiento requiere una capacidad de cómputo significativamente mayor que su homólogo reactivo (Küçük, 2012). Esto provoca que los bucles de actualización sean incapaces de ejecutarse con la frecuencia necesaria para una navegación eficaz. Una solución puede ser procesar toda la información de manera remota, en un ordenador externo con mayor potencia, y devolver los resultados de nuevo al robot. Esto es frecuente en robots de bajo coste, sin embargo debe hacer frente a otro nuevo inconveniente: los retardos en la comunicación. Esto es solventado en muchos casos mediante el aumento de los márgenes de tolerancia o, en casos de retardo de procesamiento mayor, a través de predictores. Este es el caso descrito en (Koay y Bugmann, 2004), en el que se plantea el uso de predictores de Smith para solventar desfases en el procesamiento de la navegación autónoma mediante realimentación visual.

2.3.3. Algoritmos de planificación de rutas

En lo que a algoritmos de navegación se refiere, existe una vasta literatura desde finales de la década de los 70, con la configuración espacial (*C-space*) en (Lozano-Pérez y Wesley, 1979) como primeros avances en este campo. En (Patle et al., 2019; Pol y

Murugan, 2015; Raja y Pugazhenthi, 2012) se realiza un detallado trabajo de revisión sobre los principales algoritmos de navegación autónoma, de los cuales se revisarán los más significativos. Con el objetivo de ofrecer una visión completa y estructurada de los principales enfoques, se clasifican estos algoritmos en clásicos y evolutivos.

En los albores de la navegación en robots móviles, estos algoritmos clásicos ganaron una gran popularidad a pesar de las limitaciones técnicas de la época. Uno de los puntos débiles era el gran coste computacional y la vulnerabilidad ante incertidumbres del entorno. En este sentido, proliferaron las estrategias de, junto con sus siglas en inglés, Descomposición de Celdas (CD), Hoja de Ruta (RA) y Campo Potencial Artificial (APF).

La idea básica de la descomposición de celdas es discriminar entre las áreas geométricas o celdas que están libres y las áreas que están ocupadas por obstáculos (Mohanty y Parhi, 2013). De esta forma, se divide la región en cuadrículas no superpuestas y utiliza grafos de conectividad para pasar de una celda a otra. Durante el desplazamiento, se considera que las celdas sin obstáculos (celdas puras) logran la planificación del recorrido desde la posición inicial hasta la posición final. La secuencia de celdas puras que comunica estas posiciones muestra el recorrido calculado. Este método se clasifica como adaptativo, aproximado y exacto. Un aspecto importante del método de descomposición de celdas es la definición de límites entre las celdas. Si los límites se colocan en función de la estructura del entorno, de manera que la descomposición no tenga pérdidas, entonces el método se denomina descomposición de celdas exacta (Sleumer y Tschichold-Gürmann, 1999). Si la descomposición da lugar a una aproximación del mapa real, el sistema se denomina método de descomposición de celdas aproximada (Cai y Ferrari, 2009).

En la metodología de la hoja de ruta, el planificador se encarga de conectar los puntos inicial y final mediante segmentos rectos o curvos. Una vez creado este entramado de caminos, se interconectan entre sí para la planificación de movimientos del robot a un nivel inferior. La planificación de trayectorias se reduce, por tanto, a conectar las posiciones inicial y final del robot encadenando una serie de estos caminos. Dos métodos para realizarlo son mediante grafos de visibilidad y diagramas de Voronoi. En el gráfico de visibilidad, la trayectoria del robot móvil discurre más próxima al obstáculo, consiguiendo una trayectoria de distancia mínima, mientras que en la trayectoria del diagrama Voronoi el robot móvil se mantiene lo más alejado posible de ellos. Para conseguir la ruta más corta, existen dos algoritmos fundamentales que, debido a su popularidad y efectividad, serán detallados a continuación.

Algoritmo de Dijkstra, concebido originalmente por Edsger Dijkstra en 1956 y publicado en (Dijkstra et al., 1959). Su formulación es la siguiente (LaValle, 2006): supóngase que para cada arista, $e \in E$, en la representación mediante grafos de un problema de planificación tiene un coste no negativo $l(e)$, que es el coste de aplicar una acción. El coste $l(e)$ puede ser expresado usando la representación de espacio de estados como $l(x, u)$, siendo u la acción aplicada en el estado x. El coste total del plan es la suma de los costes de las aristas a lo largo del recorrido desde el estado inicial hasta el final.

La cola de prioridad, Q, será ordenada según la función $C : X \rightarrow [0, \infty]$, denominada coste siguiente (*cost-to-come*). Para cada estado x, se define el valor $C^{*}(x)$ como el coste siguiente óptimo para el estado inicial x_I. Este coste óptimo se obtiene sumando los costes de cada arista a lo largo de todos los recorridos posibles desde x_I hasta x y escogiendo aquel con un menor coste acumulado.

Algoritmo A estrella (A^{*}) (Szekeres y Wilf, 1968). Se trata de una extensión del algoritmo de Dijkstra que intenta reducir el número total de estados explorados incorporando un estimador heurístico del coste que supondría llegar al objetivo desde un estado dado (LaValle, 2006). Sea $C(x)$ el coste siguiente desde x_I hasta x, y $G(x)$ el coste de desplazamiento (*cost-to-go*) desde x hasta algún estado X_G. Sugóngase que el coste es el número total de pasos del recorrido. Si un estado tiene coordenadas (i, j) y el siguiente (i', j'), entonces $|i' - i| + |j' - j|$ es un buen estimador, ya que es la distancia entre ambos en línea recta, ignorando los obstáculos. Si se incluyeran, este coste aumentaría por el hecho de esquivarlos. El objetivo es calcular una estimación lo más cercana posible al coste de desplazamiento sin superarlo, denotando dicha estimación con $\hat{G}^{*}(x)$.

El algoritmo A^{*} funciona de forma similar al de Dijkstra, diferenciándose en la función usada en la ordenación de Q. En A^{*}, se emplea la suma $C^{*}(x') + \hat{G}^{*}(x')$, lo que supone que la cola de prioridad se ordena mediante estimaciones del coste óptimo desde x_I hasta X_G. Si $\hat{G}^{*}(x)$ es un estimador del verdadero coste de desplazamiento óptimo para todo $x \in X$, el algoritmo A^{*} garantiza encontrar el plan óptimo. Cuanto más cercano sea $\hat{G}^{*}$ a G^{*}, menor número de vértices serán explorados en comparación con el algoritmo de Dijkstra.

Por último, el método de Campo Potencial Artificial (APF) fue desarrollado originalmente en (Khatib, 1985). El concepto es crear un camino entre las posiciones iniciales y finales mediante unas ”fuerzas artificiales” introducidas por los obstáculos y el destino. Las fuerzas generadas por los obstáculos serán repulsivas mientras que el destino incorporará una fuerza atractiva. Esto motivaría al robot a desplazarse hacia el destino a la vez que es repelido por los obstáculos de su recorrido. Un ejemplo de

aplicación de estas técnicas es la propuesta de exploración planetaria presentada en (Massari, Giardini y Bernelli-Zazzera, 2004). Una de las principales desventajas de este método son las situaciones en las que el robot se queda atrapado debido a la presencia de mínimos locales. Una solución a este problema es la propuesta en (Borenstein y Koren, 1989) mediante detección de mínimos locales y seguimiento de paredes (*wall-following method*.

Una vez revisados los principales métodos clásicos, se abordarán los llamados métodos evolutivos. En este apartado se tratarán los algoritmos genéticos, de lógica difusa, redes neuronales, optimización por enjambre de partículas.

En los algoritmos genéticos, a cada miembro de la población (diferentes individuos caracterizados por sus genes) se le asigna un valor de aptitud según la función objetivo. Estos individuos son seleccionados por dicho valor y sus genes se pasan a la siguiente generación mediante cruce. Esta mutación mantiene la diversidad de la población y previene una convergencia prematura. El algoritmo concluye cuando la población alcanza una convergencia hacia el destino especificado (Patle et al., 2019). Algunos de los estudios sobre la aplicación de la genética a la navegación autónoma se encuentra en (Kala et al., 2009; Manikas, Ashenayi y Wainwright, 2007).

La lógica difusa en un concepto acuñado originalmente en (Zadeh, 1965) con una gran repercusión y penetración en distintos campos de la investigación. Tal y como se describe por su autor, *"La lógica difusa puede ser vista como una extensión de la lógica multivalente. Sin embargo, sus usos y objetivos son algo distintos. Asimismo, el hecho de que la lógica difusa se ocupe de modos de razonamiento aproximados en lugar de precisos implica que, en general, las cadenas de razonamiento sean de corta extensión, y el rigor no juega un papel tan importante como en los sistema de lógica clásica. En resumen, en lógica difusa todo, incluida la verdad, es una cuestión de grado."* - (Zadeh, 1988).

Esta metodología se emplea en situaciones con un alto nivel de incertidumbre, complejidad y no linealidad, como pueden ser el reconocimiento de patrones, el control automático o la toma de decisiones. La hipótesis de la lógica difusa está motivada por la notable habilidad humana para procesar la información basada en percepción. Emplea una serie de normas proporcionadas por los humanos (*if - then*) y las convierte en equivalentes matemáticos. En (Seraji y Howard, 2002) se realiza una evaluación del terreno mediante la información percibida por sus cámaras a bordo para detectar su rugosidad, inclinación y discontinuidad. Con esta información y mediante reglas difusas, se obtiene un índice de viabilidad que cuantifica la facilidad para atravesar ciertas rutas.

Las redes neuronales (NN) artificiales son sistemas inteligentes que consisten en un gran número de procesos simples e interconectados. Estructuralmente, una red

neuronal es un conjunto de capas de neuronas (o nodos) conectadas entre sí. Entre ellas, se distinguen la capa de entrada (*input layer*) que recibe la información (en este caso puede ser del entorno percibido por el robot), una o varias capas ocultas (*hidden layers*) interconectadas entre sí y con la capa de entrada, y finalmente una capa de salida, que devuelve la información procesada o acción a realizar. Estos nodos computan una función de activación, que no es más que un mapeo de los datos de entrada a través de cierta función, como las funciones sigmoidal, tangente hiperbólica, ReLU, o exponencial, entre las más usadas (Karlik y Olgac, 2011). La forma de interconectar estos nodos es mediante una serie de pesos que ponderan cada valor de entrada y que se calculan, generalmente, por propagación hacia atrás (*backpropagation*). Con esta técnica se calculan los errores de salida mediante su diferencia con los valores reales y se reajustan los pesos hacia atrás desde la capa final hasta la inicial mediante gradientes para minimizar la función de coste. Un mayor detalle de la teoría de la propagación hacia atrás se detalla en (Hecht-Nielsen, 1992). En relación al apartado siguiente, (Gudise y Venayagamoorthy, 2003) compara el rendimiento de la optimización por enjambre de partículas y la propagación hacia atrás como método de cálculo de redes neuronales.

La optimización por enjambre de partículas (PSO) en un enfoque metaheurístico que trata de imitar el comportamiento de los bancos de peces o de las bandadas de pájaros, en el que todo el grupo se dirige hacia un objetivo sin la presencia de un líder, sino siguiendo a aquel que esté más cerca de lograrlo. Para ello, se requiere una buena comunicación entre todos los miembros del grupo. El algoritmo PSO permite optimizar un problema a partir de una población de soluciones candidatas, moviendo cada una de sus partículas por todo el espacio de búsqueda. El movimiento de cada uno de ellos se ve influenciado por la mejor posición anterior, así como por las mejores posiciones encontradas por toda la población. El algoritmo se encarga de encontrar un solución óptima o próxima a ella mediante la función de aptitud (*fitness function*) $f(x) = f(x_1, x_2, ..., x_n)$, en la que x_i es una población de partículas. Debido a su naturaleza metaheurística, no asegura la obtención de una solución óptima global. En (Ahmadzadeh y Ghanavati, 2012) se emplea esta metodología para esquivar obstáculos estáticos y dinámicos. Una modificación de este algoritmo es el PSO modificado (MPSO) propuesto en (Shiltagh y Jalal, 2013) en el que se introduce un factor de error para evitar la convergencia prematura propia de estos métodos y solventar el problema de planificaciones no realizables.

Además del cálculo de la ruta a seguir por el robot, un paso más allá es el realizado por los suavizadores de trayectoria. El papel de estos es la adecuación de los segmentos de camino que debe recorrer el robot para evitar discontinuidades y saltos bruscos en

el movimiento. En (Ravankar et al., 2018) se muestra un amplio abanico de suavizadores empleados en estos casos.

Algunos de estos se basan en métodos de interpolación, como la interpolación polinómica (Huh y Chang, 2014), curvas bézier (Choi, Curry y Elkaim, 2008) o curvas B-spline (Berglund et al., 2009). Igualmente, existen métodos de optimización para conseguir tales fines, como el descrito en (Zhu, Schmerling y Pavone, 2015) mediante algoritmos heurísticos o las bandas elásticas temporales (TEB) (Rösmann, Hoffmann y Bertram, 2015). Este último optimiza la trayectoria del robot respecto al tiempo de ejecución, margen con obstáculos y restricciones dinámicas del robot.

2.4. Sistemas multi-robot

Un sistema multi-robot (MRS) es un conjunto de robots diseñados para conseguir un comportamiento colectivo. Mediante este comportamiento colectivo, algunos objetivos que resultarían imposibles para un solo robot se vuelven factibles. Esta línea de investigación se remonta a la década de los ochenta, con proyectos como ALLIANCE (Parker, 1994), CEBOT (Fukuda y Kawauchi, 1990), o MURDOCH (Gerkey y Matarić, 2000). Uno de los principales retos a superar era el diseño de estrategias de coordinación que permitieran un cierto rendimiento y eficiencia en términos de tiempo y espacio de trabajo. En la última década, el estudio de los sistemas multi-robot ha crecido significativamente. Una de las razones de este auge son los beneficios inherentes de sistemas con múltiples agentes, en comparación con los de un único robot. Algunas de estas ventajas son (A. Khamis, Hussein y Elmogy, 2015):

- Complejidad en la resolución de tareas: algunos objetivos pueden ser ciertamente complejos o incluso imposibles para un único robot. Esta dificultad puede derivarse de la naturaleza de dichas tareas, como el transporte de un gran número de material, o por la diversidad de requerimientos, sirviendo de ejemplo la distinta funcionalidad que pueden ofrecer un robot móvil y un robot manipulador.

- Simplicidad del diseño: tener robots simples y de menor tamaño es más sencillo y barato de implementar en muchos casos que un único robot más potente y complejo.

- Incremento del rendimiento: el tiempo dedicado en completar las tareas puede decrementar de manera significativa, sobre todo en aquellos casos en los que se trate de procesos altamente paralelizables.

- Incremento de la fiabilidad, escalabidad y flexibilidad: el hecho de contar con más de un robot aumenta la fiabilidad por la redundancia de recursos, ya que un solo robot puede provocar un cuello de botella en todo el sistema en casos críticos. Por ejemplificarlo, si una tarea la realizan varios agentes, el sistema de vuelve más tolerante a fallos, puesto que en caso de inoperatividad de unos de ellos, el resto puede ocuparse de completar sus funciones.

Para poder implementar un sistema multi-robot en el mundo real, es necesario resolver una serie de problemas. Algunos de los más recurrentes son la asignación de tareas, la formación en grupo, la detección y seguimiento cooperativo de objetos, la evitación de obstáculos o la comunicación entre los distintos robots.

Uno de los que mayor repercusión tiene en el rendimiento del sistema es la asignación multi-robot de tareas (MRTA). Los enfoques en los que se basan los algoritmos para su resolución son los basados en mercado o en optimización. Los basados en mercado, mayoritariamente mediante esquemas de subasta, muestran una mayor eficiencia, robustez y escalabilidad, como demuestran los trabajos (Dias y Stentz, 2002; A. M. Khamis, Elmogy y Karray, 2011). En contrapartida, los basados en optimización ofrecen un rendimiento mayor a cambio de un incremento del coste computacional. Esto es asumible en casos en los que el número de agentes es reducido pero es difícilmente escalable para sistemas de mayor dimensión. Los algoritmos genéticos (Jones, Dias y Stentz, 2011) o los de colonia de hormigas (*ant colony optimization*) (Wang, Gu y Li, 2012) son algunas muestras.

La forma de abordarlos, así como la toma de decisiones puede ser desde una perspectiva centralizada, en la que un procesador central distribuye las tareas y comunica a cada agente el estado de los demás, o descentralizada, si es cada robot el encargado de llevar a cabo sus funciones sin más información que la que le proporcionan sus sensores.

Una arquitectura centralizada procesa una información global del entorno y del estado de todos los robots y la comparte con todos ellos. Con ello, se consigue un rendimiento óptimo, por ejemplo, en la generación de trayectorias en el caso de robots móviles. Sin embargo, esta arquitectura sufre una serie de limitaciones. En primer lugar, es sólo útil cuando el número de robots es relativamente pequeño, ya que en grandes sistemas el coste computacional sería prohibitivo. Por otro lado, la robustez del sistema se vería comprometida si existe un fallo de comunicación o problemas con el agente central, ya que todos los robots quedarían inoperativos. Algunas muestras de esta arquitectura serían (Yamashita et al., 2003), (Clark, Rock y Latombe, 2003) o (Liu y Kroll, 2012).

En la perspectiva descentralizada no existe ese ordenador central, por lo que todos los robots son iguales y autónomos en la toma de decisiones. En este tipo de sistemas la respuesta frente a cambios dinámicos o incertidumbres suele ser mejor y ofrece una mayor flexibilidad, versatilidad y robustez. Como contraprestación, el rendimiento no será el óptimo, ya que cada robot su conocimiento del mundo es limitado. (Prorok, Bahr y Martinoli, 2012) y (Montemayor y Wen, 2005) son ejemplos de sistemas colaborativos descentralizados.

Capítulo 3

Materiales y métodos

En este capítulo se presentan los materiales y dispositivos involucrados en la construcción del prototipo robot con el que se ha diseñado el sistema. Asimismo, se muestra la implementación de algunos de los algoritmos descritos en el capítulo 2 mediante sus correspondientes paquetes ROS o código propio.

3.1. Arquitectura hardware del robot

En el presente trabajo se ha realizado el prototipo con el robot Create® 2 de la marca iRobot®. Como periféricos se han añadido todos los dispositivos para la navegación, percepción y procesamiento necesarios para una correcta navegación.

Todos las características técnicas del robot, así como de los componentes adicionales serán detallados en los siguientes apartados.

3.1.1. iRobot® Create® 2

Se trata de un robot de tracción diferencial con batería recargable de 14,4 V y 3000 mAh de capacidad. Su peso es de unos 3,5 kilogramos y sus dimensiones, en milímetros, según su hoja de especificaciones (iRobot, 2018) se muestran en la figura 3.1.1.1.

Cuenta con diversos sensores integrados para la percepción de su entorno.

- Sensores infrarrojos: dispone de un total de 6 sensores horizontales para detectar obstáculos frontales y laterales y 4 sensores verticales con el objetivo de avisar de la proximidad a un precipicio (figura 3.1.1.2).

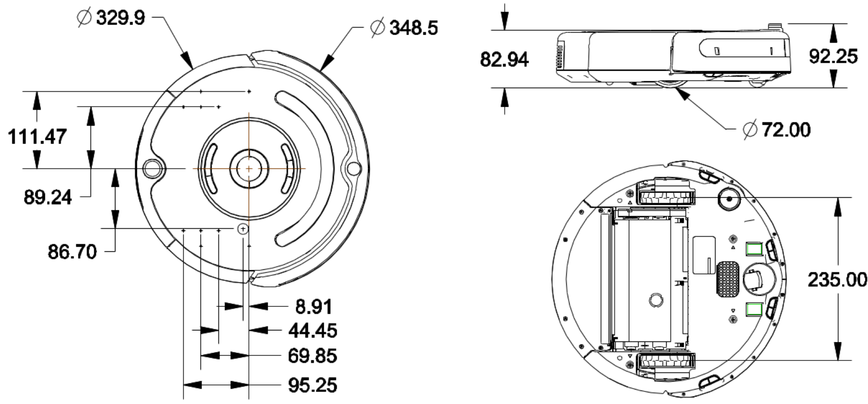


Figura 3.1.1.1: Dimensiones iRobot Create 2

Figura 3.1.1.2: Sensores infrarrojos horizontales (superior) y verticales (inferior)

- Sensores de contacto (bumpers): uno a cada lado del frontal del robot (figura 3.1.1.3).

Figura 3.1.1.3: Sensores de contacto

- Sensores de caída de rueda: sensores binarios que indican si las ruedas se han descolgado, por ejemplo, por inclinación provocada por un cambio de nivel (figura 3.1.1.4).

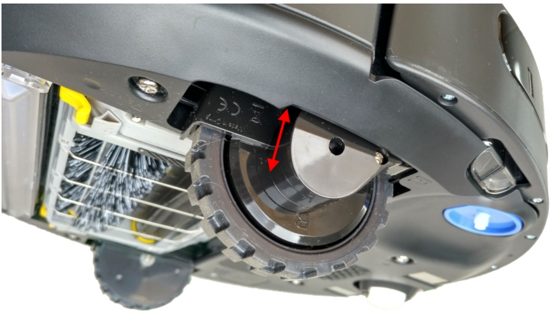


Figura 3.1.1.4: Sensores de caída de rueda

- Sensores infrarrojos de comunicación: realizan el intercambio de señales entre el robot y la estación de carga durante el acoplamiento automático (figura 3.1.1.5).

Figura 3.1.1.5: Sensores infrarrojos base - robot

- Sensores de estado de la batería: permite consultar su voltaje e intensidad de carga o descarga. Además incluye sensor de temperatura y de estado de carga.

- Encoders: uno por cada rueda, con una resolución de 508,8 cuentas por vuelta.

En cuanto a los actuadores incorporados, cuenta con:

- Motores DC: cada una de las rueda está acoplada a un motor de corriente continua (elipses rojas en la figura 3.1.1.6) que permiten al robot desplazarse en línea recta a una velocidad máxima de 0,5 m/s y describiendo radios de hasta 2 metros en trayectorias curvas.

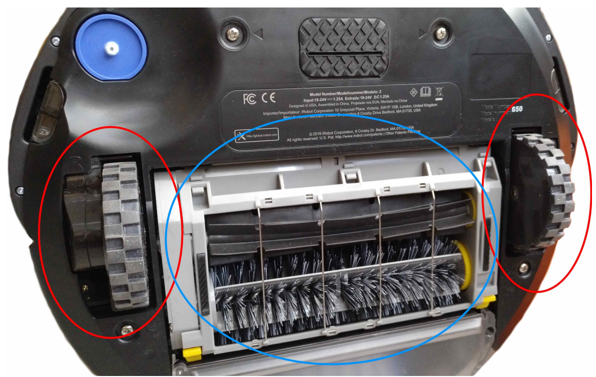


Figura 3.1.1.6: Actuadores

- Cepillos y aspirador: al igual que los robots Roomba comerciales se incluye la capacidad de aspirar y limpiar la superficie por la que se desplaza el robot (elipse azul en la figura 3.1.1.6).

- Altavoces: incorpora altavoces con los que se pueden programar alertas sonoras y con los que el robot avisa por defecto de situaciones particulares como puede ser el caso de batería baja.

- LEDs: en la parte superior se encuentran una serie de LEDs, iguamente programables para mostrar caracteres al usuario.

De manera predeterminada cuenta con cuatro modos automáticos de funcionamiento.

Modo apagado

Después un cambio de batería o tras el primer encendido, el robot se queda en un estado apagado a la espera del comando ”Start”. Una vez recibido, puede entrar en cualquiera de los cuatro modos operativos.

Modo pasivo

En modo pasivo, se puede solicitar y recibir datos sobre los sensores pero no se puede modificar el estado de los actuadores. Para ello, se requiere cambiar a los modos seguro o completo

Modo seguro

Este modo permite interactuar con el robot con una mayor libertad, pudiendo dar órdenes sobre el movimiento siempre y cuando no se violen las siguientes condiciones de seguridad:

- Detección de un precipicio mientras se mueve hacia delante o en giro con un radio inferior al radio de robot.
- Detección de caída de una rueda.
- Conexión del cargador.

Cuando se da una de las condiciones anteriores, el robot regresa al modo pasivo.

Modo completo

Confiere un control total sobre el robot, desactivando todas las reglas de seguridad.

3.1.2. Raspberry Pi Model 3B+

El popular microordenador desarrollado por Raspberry Pi ® (figura 3.1.2.1) es uno de los más usados en proyectos del mundo *maker* y de la robótica debido a su versatilidad y gran soporte entre su comunidad. Las especificaciones de la tercera revisión de este modelo, según su fabricante (Raspberry Pi Foundation, 2020b), son las siguientes:

Figura 3.1.2.1: Raspberry Pi Model 3B+

- Procesador Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC con una frecuencia de reloj de 1,4 GHz.
- Memoria RAM LPDDR2 SDRAM de 1GB.
- Conectividad LAN inalámbrica 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac, Bluetooth 4.2, BLE.
- Gigabit Ethernet sobre USB 2.0 (tasa máxima de transferencia de 300 Mbps).
- 40 pines GPIO (entrada - salida).
- Puerto HDMI.
- 4 puertos USB 2.0.
- Puerto CSI para cámara Raspberry Pi.
- Puerto DSI para display táctil Raspberry Pi.
- Puerto de salida estéreo de 4 polos y vídeo compuesto.
- Puerto Micro SD.
- Entrada de alimentación 5V / 2,5A.
- Soporte para alimentación sobre Ethernet (PoE).

3.1.3. Slamtec RPLIDAR A3M1

Un *lidar* (acrónimo del inglés LIDAR, Laser Imaging Detection And Ranging) es un dispositivo que permite obtener la distancia comprendida entre el punto del emisor y un

cuerpo o superficie lejana. Esta distancia se calcula midiendo el desfase comprendido entre la emisión del haz láser y la recepción de la señal reflejada. La tecnología lidar tiene aplicaciones en geología, sismología y robótica, especialmente en robótica móvil. Un uso muy recurrente en los últimos años es su incorporación en prototipos de vehículos autónomos.

A grandes rasgos, los lidar pueden ser de dos tipos: mecánicos y sólidos. En el primer caso, emplean un engranaje móvil para crear un amplio campo de visión. Sin embargo, lastran un gran ratio señal-ruido y su construcción es bastante voluminosa. En cambio, los lidar de estado sólido no tienen componentes móviles, reduciendo asi el ruido. Este tipo, no obstante reduce también su campo de visión, por lo que se suelen emplear múltiples canales para poder competir en este aspecto con los mecánicos.

En el caso del presente trabajo, el dispositivo empleado es el RPLIDAR A3M1 de la firma Slamtec (figura 3.1.3.1).

Figura 3.1.3.1: Slamtec RPLIDAR A3M1

Su hoja técnica proporcionada por el fabricante (Slamtec, 2019) muestra su alcance, resolución y principales características (figura 3.1.3.2)

El funcionamiento del este dispositivo se basa en el principio de triangulación láser (figura 3.1.3.3) e incorpora un sistema de visión y adquisición de datos de alta velocidad con el que consigue una frecuencia de muestreo de 16000 muestras por segundo.

Un aspecto relevante, sobre todo a la hora de su disposición en el robot es su ventana óptica (figura 3.1.3.4). Para que el lidar ofrezca todo su rango de detección es básico que ésta quede libre de obstáculos, como podrían ser el resto de los periféricos montados.

Item	Enhanced Mode	Outdoor Mode
Application Scenarios	Extreme performance Ideal for indoor environments with maximum ranging distance and sampling frequency.	Extreme reliability Ideal for both outdoor and indoor environments with reliable resistance to daylight.
Operating Range	White object: 25 meters	White object: 20 meters
	Black object: 10 meters	Black object: TBD
Minimum Operating ranging	0.2m	0.2m
Sample Rate	16 kHz	16 kHz or 10 kHz
Scan Rate	Typical value: 10 Hz (adjustable between 5 Hz-15 Hz)	
Angular Resolution	0.225°	0.225° or 0.36°
Scan Field Flatness	±1.5	
Communication Interface	TTL UART	
Communication Speed	256000 bps	
Compatibility	Support former SDK protocols	

Figura 3.1.3.2: Especificaciones RPLIDAR A3M1

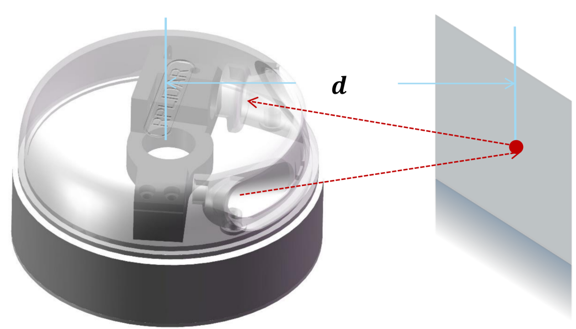


Figura 3.1.3.3: Triangulación láser RPLIDAR A3M1

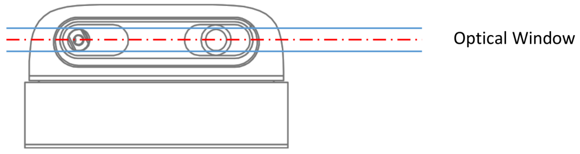


Figura 3.1.3.4: Ventana óptica RPLIDAR A3M1

3.1.4. Batería LiPo 11,1V

Debido a la escasa potencia de la salida externa de la batería integrada del Create 2, es necesario acoplar una segunda batería para alimentar a la Raspberry Pi y el lidar. Como ya se describió en el apartado 3.1.2, se recomienda una corriente de alimentación de unos 2,5 amperios, aunque en la práctica puede ser algo menor. A su vez, la intensidad máxima demandada por el lidar (Slamtec, 2019) es de 1,5 amperios en el arranque, lo cual suman una corriente máxima teórica de 4 amperios. Sin embargo, dado que nunca van a demandar el pico de corriente simultáneamente y el consumo real de la raspberry suele encontrarse en torno a 1 amperio, una batería que consiga ofrecer unos 3 amperios sería más que suficiente.

La batería seleccionada finalmente ha sido la U-TECH PRO LiPo de 3 celdas (11,1V) y de 2200 mAh de capacidad (U-Tech, 2020), mostrada en la figura 3.1.4.1. Su corriente nominal máxima es de 66 amperios, por lo que cumple de largo los requerimientos del proyecto.

Figura 3.1.4.1: Batería LiPo U-TECH 2200 mAh

3.1.5. Convertidor buck DC-DC

Para adaptar los niveles de tensión es necesario un convertidor buck DC-DC reductor (figura 3.1.5.1), como el convertidor ajustable de 75W basado en el XL4015 (HWAYEH, 2020). Su tensión de entrada varía entre los 4 y los 38 voltios, con una salida entre los 1,25 y los 36 voltios.

Con todos estos periféricos instalados, el robot final se presentará en el capítulo 4.

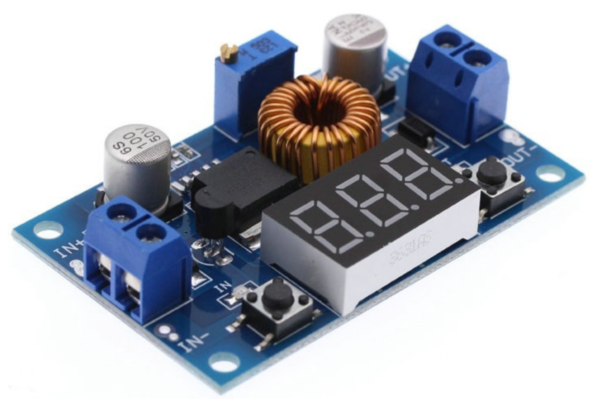


Figura 3.1.5.1: Convertidor buck DC-DC

3.2. Arquitectura software del robot

3.2.1. Sistema Operativo

Una computadora moderna (Tanenbaum y Bos, 2015) consta de uno o más procesadores, memoria principal, discos, impresoras, un teclado, un ratón, una pantalla, interfaces de red y varios otros dispositivos de entrada/salida. En general, un sistema complejo. Si todos los programadores de aplicaciones tuvieran que comprender cómo funcionan todas estas cosas en detalle, nunca se escribiría ningún código. Además, administrar todos estos componentes y usarlos de manera óptima es un trabajo extremadamente desafiante. Por esta razón, las computadoras están equipadas con una capa de software llamada sistema operativo, cuyo trabajo es proporcionar a los programas de los usuarios un modelo de la computadora mejor, más simple y más limpio, y manejar la administración de todos los recursos que se acaban de mencionar figura 3.2.1.1. El sistema operativo, la pieza fundamental del software, funciona en modo kernel (también llamado modo supervisor). En este modo tiene acceso completo a todo el hardware y puede ejecutar cualquier instrucción de la máquina. El resto del software funciona en modo usuario, en el cual sólo está disponible un subconjunto de todas las instrucciones.

Es difícil precisar qué es un sistema operativo aparte de decir que es el software que se ejecuta en modo kernel, e incluso eso no siempre es cierto. Parte del problema es que los sistemas operativos realizan esencialmente dos funciones: proporcionar a los programas un conjunto abstracto y limpio de recursos y administrar estos recursos hardware. Esta abstracción es la clave para gestionar toda esta complejidad. Las buenas abstracciones convierten una tarea casi imposible en dos manejables. El primero es

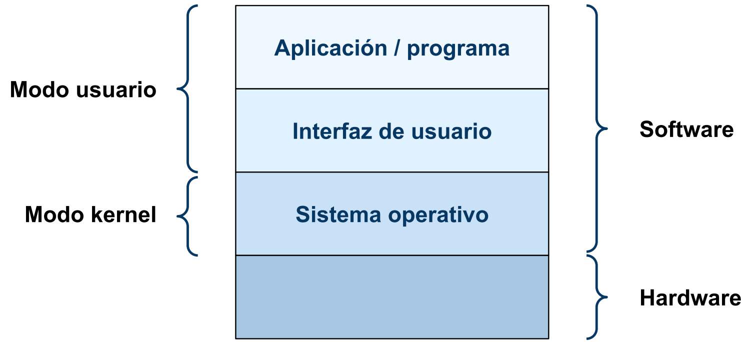


Figura 3.2.1.1: Nivel de abstracción del sistema operativo

definir e implementar las abstracciones. El segundo es usar estas abstracciones para resolver el problema en cuestión.

En el caso que nos ocupa, la elección del sistema operativo está supeditada a la compatibilidad con el entorno de desarrollo que se empleará en el robot, el cual se detallará en el apartado 3.2.2. Por esta razón, un candidato relativamente reciente (lanzamiento el 26 de abril de 2018) es el Ubuntu 18.04 LTS Bionic Beaver (Canonical Ltd., 2020b) para nuestra unidad central.

Para las Raspberry de las unidades móviles las alternativas están menos estandarizadas, ya que deben tener en cuenta las limitaciones del dispositivo. Se han probado las siguientes distribuciones de linux:

- NOOBS (New Out Of the Box Software) (Raspberry Pi Foundation, 2020a): sus principales ventajas son una instalación sencilla y rápida para los usuarios más novatos. Sin embargo, es una distribución muy capada con demasiadas limitaciones que impiden poder sacar todo el partido a la placa.

- Ubuntu Mate 18.04 (Canonical Ltd., 2020c): un sistema operativo potente, que ofrecería una buena experiencia de usuario en un equipo de sobremesa, pero que para los ordenadores monoplaca es demasiado pesado.

- Ubuntu 18.04 Server (Canonical Ltd., 2020a): ésta ha sido una alternativa con la que se ha trabajado en el desarrollo del prototipo, ya que incluye repositorios para instalar la mayoría de paquetes de ROS de forma precompilada, lo cual evita los largos tiempos de espera que conlleva compilar en una máquina de poca potencia. Su instalación básica no incluye entorno de escritorio, por lo que han probado diferentes versiones, como XFCE o LXDE. A pesar de ello, diversos problemas a la hora de ejecutar los programas obligaron al cambio hacia Raspbian.

- Raspbian Buster (Raspberry Pi Foundation, 2020c): esta distribución creada específicamente para Raspberry y basada en Debian 10, fue inicialmente descartada debido a la falta de ciertos repositorios de paquetes de ROS precompilados y a la necesidad de compilarlos de forma local. No obstante, gracias a su fluidez y mejor interfaz para acceder de forma remota, es el sistema operativo escogido.

3.2.2. Robot Operating System (ROS)

ROS (Quigley, Gerkey y Smart, 2015) es un entorno de desarrollo para diseñar software aplicado a la robótica. Es una colección de herramientas, bibliotecas y convenciones que tiene como objetivo conseguir un rendimiento robusto y complejo en una amplia variedad de plataformas robóticas. A veces se le denomina meta-sistema operativo porque realiza muchas funciones de un sistema operativo (Fairchild y Harman, 2017), pero requiere el sistema operativo de una computadora como Linux. Uno de sus propósitos principales es proporcionar comunicación entre el usuario, el sistema operativo de la computadora y el equipo externo. Este equipo puede incluir sensores, cámaras y robots. Como con cualquier sistema operativo, el beneficio de ROS es la abstracción de hardware y su capacidad para controlar un robot sin que el usuario tenga que conocer todos sus detalles.

3.2.2.1. Historia

ROS (Quigley, Gerkey y Smart, 2015) es un gran proyecto que tiene muchos antepasados y contribuyentes. A comienzos de siglo, la comunidad de investigación en robótica ponía el acento en la necesidad de un marco de colaboración abierto. Varios proyectos en la Universidad de Stanford a mediados de la década de 2000 que incluían una IA integradora e incorporada, como el programa STanford AI Robot (STAIR) (Quigley, Berger y Ng, 2007) y Personal Robots (PR) (Wyobek et al., 2008), crearon prototipos internos de los tipos de sistemas de software flexibles y dinámicos descritos en este libro. En 2007, Willow Garage, Inc., una incubadora de robótica cercana, proporcionó recursos significativos para ampliar estos conceptos mucho más y crear implementaciones suficientemente bien probadas. El esfuerzo fue impulsado por innumerables investigadores que contribuyeron con su tiempo y experiencia al núcleo de ROS y sus paquetes de software fundamentales. En todo momento, el software se desarrolló de manera libre utilizando la licencia permisiva de código abierto BSD, y gradualmente se hizo ampliamente utilizado en la comunidad científica.

ROS (Quigley et al., 2009) fue diseñado para hacer frente al conjunto específico de desafíos encontrados al desarrollar robots de servicio a gran escala como parte de dichos proyectos, pero la arquitectura resultante es mucho más genérica que los dominios de la robótica de servicios y de manipulación. Los objetivos esenciales de ROS pueden resumirse como:

- Punto a punto (Peer-to-peer)
- Multilenguaje
- Basado en herramientas
- Ligero
- Libre y de código abierto

Punto a punto (Peer-to-Peer)

Un sistema creado con ROS consta de una serie de procesos, potencialmente en varios hosts diferentes, conectados en tiempo de ejecución en una topología punto a punto. Aunque los marcos basados en un servidor central (p. Ej., CARMEN (Montemerlo, Roy y Thrun, 2003)) también pueden conseguir los beneficios del diseño multiproceso y de múltiples hosts, un servidor de datos central puede ser problemático si las computadoras están conectadas en una red heterogénea. Por ejemplo, en los grandes robots de servicio para los que se diseñó ROS, generalmente hay varias computadoras a bordo conectadas a través de Ethernet. Este segmento de red se conecta a través de LAN inalámbrica a máquinas externas de alta potencia que ejecutan tareas de computación intensiva como la visión por computador o el reconocimiento de voz. Ejecutar el servidor central ya sea a bordo o de forma remota resultaría en un innecesario tráfico de red a través del enlace inalámbrico.

En contraposición, la conectividad punto a punto, en combinación con el correspondiente software *fanout* cuando sea necesario, evitan por completo este problema. La topología peer-to-peer requiere un mecanismo de consulta para permitir que los procesos se encuentren unos a otros en tiempo de ejecución. Esto se denomina servicio de nombre (*name service*) o *master*.

Multilenguaje

Al escribir código, cada programador tiene sus preferencias por algunos lenguajes de programación en lugar de otros. Estas preferencias son el resultado de un balance

entre el tiempo de programación, la facilidad de depuración, la sintaxis, la eficiencia del tiempo de ejecución y otras razones, tanto técnicas como profesionales. Por ello, ROS se diseña para que sea neutral en cuanto al lenguaje. Actualmente, admite cuatro lenguajes muy diferentes: C++, Python y LISP, con soporte para otros lenguajes en diversos grados de compatibilidad. La especificación de ROS se encuentra en la capa de mensaje. La configuración de la conexión entre pares se realiza en XML-RPC, para lo cual existen implementaciones robustas en la mayoría de los principales lenguajes. En lugar de proporcionar una implementación basada en C con interfaces *stub* (implementación sin finalizar) generadas para todos los idiomas principales, ROS está implementado de forma nativa en cada lenguaje de destino, para seguir las convenciones de cada uno de ellos.

Basado en herramientas

En un esfuerzo por gestionar la complejidad de ROS, está diseñado según una estructura de microkernel, donde se usa una gran cantidad de pequeñas herramientas para construir y ejecutar los diversos componentes de ROS en lugar de construir un entorno de desarrollo y ejecución monolítico. Estas herramientas realizan diversas tareas, por ejemplo, navegar por el árbol del código fuente, obtener y establecer parámetros de configuración, visualizar la topología de conexión punto a punto, medir la utilización del ancho de banda, trazar gráficamente los datos del mensaje, generar automáticamente la documentación del documento, etc. Aunque es posible una implementación de servicios básicos como un reloj global y un registrador (*logger*) dentro del módulo *master*, se opta por insertar todo en módulos separados. Esta pérdida de eficiencia se compensa por las ganancias en estabilidad y gestión de la complejidad.

Ligero

Tal y como se describe en (Makarenko, Brooks y Kaupp, 2007), la mayoría del software para robots contiene drivers y algoritmos que podrían ser reutilizables fuera de su proyecto original. Desgraciadamente, debido a una sinfín de razones, una gran parte de este código es tan dependiente del *middleware* que dificulta enormemente la extracción de su funcionalidad y su reutilización fuera de su contexto. Para contrarrestar esto, se recomienda que todos el desarrollo se realice en bibliotecas separadas sin dependencia de ROS y que posteriormente sea el sistema de compilación de ROS el que lleve a cabo compilaciones modulares a través de CMake. ROS utiliza código de múltiples proyectos de código libre, como la visión por

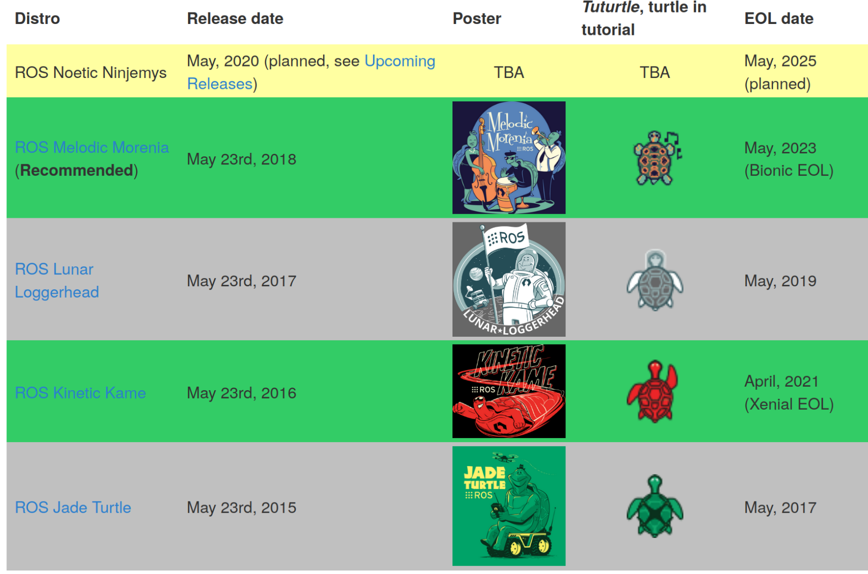
computador de OpenCV (Bradski y Kaehler, 2008) o algoritmos de planificación de OpenRAVE (Srinivasa et al., 2008).

Libre y de código abierto

Todo el código fuente de ROS está disponible de manera pública. Esto es un factor crítico a la hora de la depuración en todos los niveles de la pila de software. ROS se distribuye bajo las condiciones de la licencia BSD, que permite el desarrollo de proyectos comerciales y no comerciales.

3.2.2.2. Uso de ROS

A fecha de elaboración del proyecto, la versión más moderna de ROS es la Melodic Morenia (figura 3.2.2.2.1), por lo que ha sido la opción escogida para el desarrollo (Open Robotics, 2020d).



Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys	May, 2020 (planned, see Upcoming Releases)	TBA	TBA	May, 2025 (planned)
ROS Melodic Morenia (**Recommended**)	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017

Figura 3.2.2.2.1: Versiones ROS

De manera conceptual, y a grandes rasgos, el uso de ROS se basa en la existencia de procesos que ejecutan una tarea, llamados **nodos**, que se comunican entre sí mediante **mensajes** publicados en categorías específicas llamadas **tópicos** (*topics*). A su vez, estos nodos pueden interactuar entre sí mediante la llamada a **servicios**, en los cuales se realiza una petición por parte del proceso que demanda la ejecución de cierto programa y se recibe una respuesta por parte del nodo que la lleva a cabo. Por último, el proceso ***master*** permite que los nodos puedan encontrarse unos a otros y el

servidor de parámetros registra una serie de variables que facilitan la ejecución de los programas.

Nodos

En esencia, los nodos son procesos software que tienen la capacidad de registrarse en el nodo *master* y comunicarse con otros nodos. La idea del diseño de ROS es que cada nodo sea independiente de los demás, con el objetivo de que una tarea compleja sea descompuesta en tareas más sencillas que ejecuten cada uno de ellos.

Tópicos

La información que proporcionan los nodos es volcada al conjunto de la infraestructura de ROS a través de los tópicos. De esta manera, puede ser empleada por varios procesos para desempeñar su función. Un ejemplo de ello puede ser el tópico en el que el robot publique el escáner lidar. Este mensaje puede ser recogido por el nodo AMCL para localizar el robot en su entorno y, a su vez, procesado por el nodo de navegación autónoma para evitar obstáculos a nivel local.

Mensajes

Los mensajes se definen por tu tipo y formato. Estas definiciones pueden ser extraídas de los paquetes estándar o creadas por el usuario.

Master

Los nodos ROS suelen ser programas independientes que pueden ejecutarse de manera concurrente en varios sistemas (Fairchild y Harman, 2017). El master de ROS proporciona servicios de nombres y registros a los nodos en el sistema ROS. Realiza un seguimiento de los publicadores y suscriptores de los tópicos y posibilida la comunicación entre los nodos.

El papel del maestro es permitir que los nodos individuales se encuentren entre sí. El protocolo más utilizado para la conexión es el Protocolo de Control de Transmisión (TCP/IP), llamado TCPROS en ROS. Una vez que estos nodos pueden localizarse, pueden comunicarse entre sí de igual a igual.

El software ROS es en realidad un *middleware* creado para simplificar las comunicaciones y facilitar la escritura de módulos en el control de robots (González-Nalda et al., 2015). Además, las comunicaciones se pueden hacer seguras implementando un servidor VPN. Esta red de nodos ROS tiene las ventajas de la modularidad de la arquitectura, permitiendo adaptarla a sistemas ya implantados para mejorar su evolución deforma incremental y la redundancia gracias al esquema publicador/suscriptor de ROS.

Servicios

El modelo de publicación-suscripción es un paradigma de comunicación muy flexible, pero su transporte unidireccional ”de muchos a muchos” no es apropiado para las interacciones de solicitud/respuesta RPC (Remote Procedure Call), que a menudo se requieren en un sistema distribuido (Open Robotics, 2020f). La solicitud/respuesta se realiza a través de un servicio, que se define mediante un par de mensajes: uno para la solicitud y otro para la respuesta. Un nodo ROS proveedor ofrece un servicio, y un cliente llama al servicio enviando el mensaje de solicitud y esperando la respuesta. Las bibliotecas de clientes suelen presentar esta interfaz al programador como si fuera una llamada a procedimiento remoto.

Servidor de parámetros

Un servidor de parámetros es un diccionario multi-variable compartido al que se puede acceder a través de las API de red (Open Robotics, 2020e). Los nodos usan este servidor para almacenar y recuperar parámetros en tiempo de ejecución. Como no está diseñado para un alto rendimiento, se suele emplear para datos estáticos, como los parámetros de configuración. Está destinado a ser visible globalmente para que las herramientas puedan inspeccionar fácilmente el estado de configuración del sistema y modificarlo si es necesario.

3.2.3. Gazebo

Gazebo es un motor de simulación de físicas con soporte para ROS (Joseph, 2019), aunque tiene funcionalidad también de forma completamente independiente. Es un simulador que permite testar algoritmos y diseños de robots, realizar comprobaciones de regresión y entrenar sistemas de Inteligencia Artificial en escenarios realistas (Open Robotics, 2020c).

La mayoría de los modelos que se crean en Gazebo están en un formato XML llamado Formato de Descripción de Simulación (SDF) (Joseph, 2019). ROS tiene un enfoque diferente para representar modelos de robots. Se definen en un formato XML llamado Formato Universal de Descripción Robótica (URDF). Aún así, los modelos en URDF pueden convertirse en SDF añadiendo ciertas etiquetas.

Para lograr la integración de ROS con Gazebo, se necesitan ciertas dependencias que establezcan una conexión entre ambos y conviertan los mensajes de ROS en información comprensible de Gazebo. También se requiere un marco que implemente controladores de robot en tiempo real que ayuden al robot a moverse. El primero constituye el paquete gazebo_ros_pkgs, que es un conjunto de empaquetadores (*wrappers*) de ROS escritos para ayudar a Gazebo a comprender los mensajes y servicios ROS, mientras que el segundo constituye los paquetes ros_control y ros_controller, que proporcionan la funcionalidad para controlar los actuadores del robot y drivers listos para usar en el control de posición, velocidad o esfuerzo.

3.2.4. Interfaz con los dispositivos

3.2.4.1. iRobot ® Create ® 2

Para evitar tener que lidiar con comunicaciones serie a bajo nivel, se utiliza el paquete de ROS *create_autonomy*, que nos facilitará la interfaz con el robot (Perron, 2020a). Este es realmente una colección de cuatro paquetes que facilitan enormemente la comunicación y el control, basados en una biblioteca de C++ llamada *libcreate* (Perron, 2020b):

- ca_description: modelo URDF y mallas 3D para los robots Roomba y Create 1 y 2.
- ca_driver: controlador ROS para los robots Create 2 y Roomba.
- ca_msgs: mensajes comunes para la interfaz con el controlador.
- ca_tools: colección de herramientas para trabajar con los robots.

El principal paquete que se usa es el ca_driver, ya que inicia un nodo que publica todos los tópicos con la información que extrae de los sensores del robot y se subscribe automáticamente a aquellos sobre los que se puede publicar para transmitirle órdenes. De los múltiples tópicos que proporciona (Open Robotics, 2020a), se incluyen los esenciales que se usan.

- battery/charge_ratio: muestra el porcentaje de batería que le queda al robot.

- bumper: indica el estado de los sensores de contacto así como de los sensores infrarojos.
- joint_states: publica el estado de las articulaciones del robot, en este caso, la posición y velocidad de las ruedas directrices. Este es crucial para saber la posición y velocidad del robot.
- odom: estima la posición del marco de referencia para la odometría, es decir, la posición del robot basándose únicamente en la información que ofrecen los encoders.
- tf: publica la transformada entre todos los marcos de referencia de los elementos del robot y entre el de odometría y de la base del robot.
- cmd_vel: recibe los comandos de velocidad que debe seguir el robot.

3.2.4.2. RPLIDAR A3M1

La interfaz con el dispositivo lidar se realizar mediante un paquete ROS (Slamtec, 2020) proporcionado por el propio fabricante. Basta con incluirlo en el archivo desde el que se lanzan los nodos y configurar los espacios de trabajo y nombres de los marcos de referencia para integrarlo al robot.

3.2.5. Software de navegación

En esta sección se introducen los principales paquetes ROS y bibliotecas que se han empleado en la realización de este trabajo.

3.2.5.1. SLAM

Como ya se adelantaba en el apartado 2.1, los algoritmos SLAM permiten realizar el mapeado del entorno mientras se recalcula la posición del robot en él. La implementación en ROS ha sido posible mediante el paquete *gmapping* (Brian Gerkey, 2020b).

3.2.5.2. Localización

Siguiendo la línea que se comentaba en el apartado 2.2, la localización se ha implementado mediante el método de Monte-Carlo. En ROS, el código necesario para transferirlo al robot se lleva a cabo con el paquete AMCL (Brian Gerkey, 2020a).

3.2.5.3. Navegación autónoma

Para la navegación, se emplea el paquete *move_base*. Este paquete integra la información recopilada por los sensores a bordo del robot, junto con el mapa del entorno y la localización en él, a cargo de AMCL. Proporciona un nodo que permite configurar y ejecutar el *stack* de navegación de ROS. En su web se muestra un diagrama a alto nivel de su funcionamiento (figura 3.2.5.3.1).

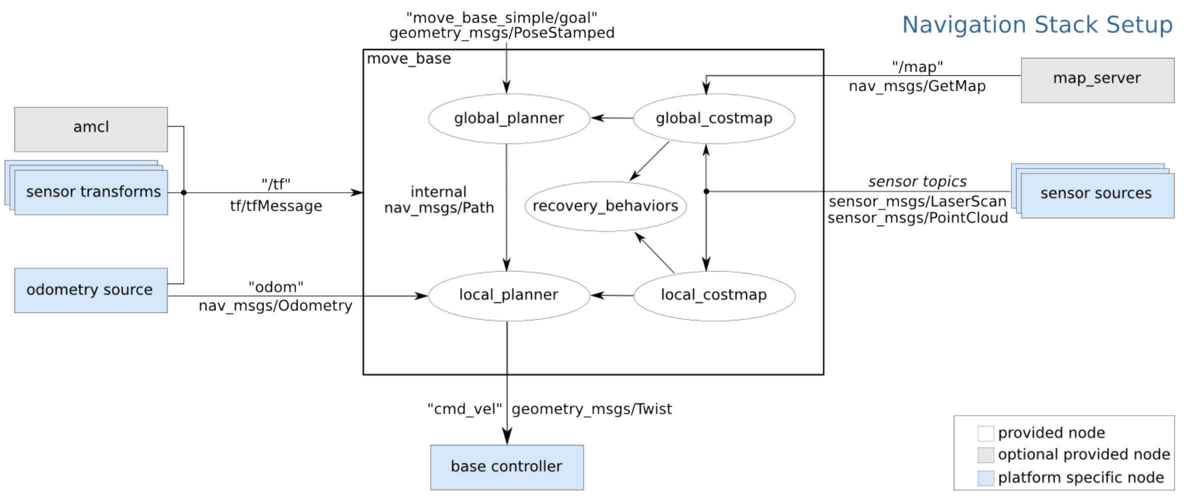


Figura 3.2.5.3.1: Diagrama del *stack* de navegación de ROS

Una vez configurado correctamente, el robot será capaz de alcanzar su destino dentro de los márgenes de tolerancia impuestos. En el caso de encontrarse atrapado en su recorrido, se realizarán movimientos de recuperación (figura 3.2.5.3.2). En primer lugar, se procede a limpiar las inmediaciones del robot del mapa en una región de espacio delimitada por el usuario. En caso de no recuperarse, se procede a rotar sin desplazamiento para limpiar el espacio contiguo. Si nuevamente falla, se reseteará el mapa de forma más agresiva, con la siguiente rotación sin desplazamiento. Si tras todos estos intentos no se consigue liberar al robot, se aborta el plan y se notifica el mensaje de error.

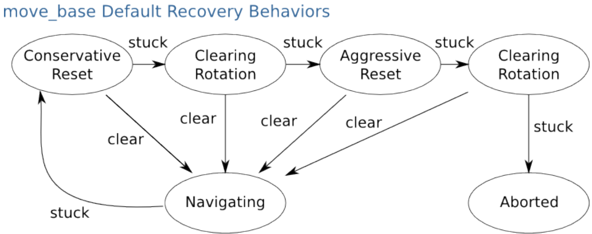


Figura 3.2.5.3.2: Diagrama de los movimientos de recuperación

De forma complementaria al paquete *move_base*, se ha empleado el paquete *nav_core*. Su función es establecer una interfaz configurable entre los planificadores globales, locales y los movimientos de recuperación. Según aparece en su web (Marder-Eppstein, 2020), su diagrama de flujo es el que se muestra en la figura 3.2.5.3.3.

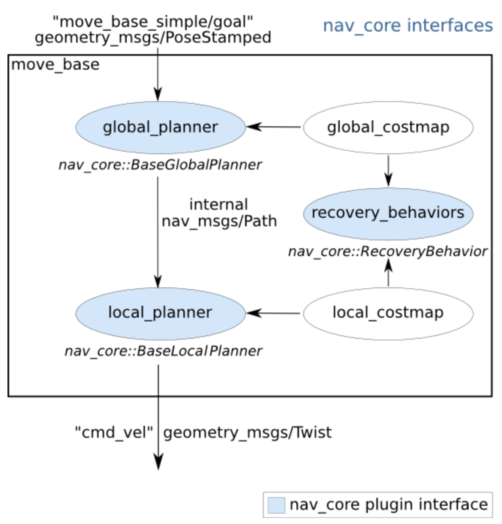


Figura 3.2.5.3.3: Diagrama de *nav_core*

Para la planificación global, se ha utilizado el paquete *global_planner* (D. Lu, 2020), el cual ejecuta los algoritmos de Dijkstra y A^*, escogiéndose el primero de ellos en este caso.

En cuanto al planificador local, el paquete *base_local_planner* (Marder-Eppstein y Perko, 2020) se ha empleado por su implementación de los algoritmos basados en DWA (*Dynamic Window Approach*) y *Trajectory Rollout*. A su vez, se ha comparado con el rendimiento del paquete *teb_local_planner* (Rösmann, 2020) basado en bandas elásticas temporales, para decidir cual implementar en el robot real.

3.2.5.4. Planificación multi-robot

Como punto final del presente proyecto se ha diseñado un mecanismo de asignación de tareas para sistemas multi-robot (MRTA). Para que todos los agentes se comporten como un sistema, deben ser capaces de distribuirse de manera óptima las tareas a realizar, con el propósito de reducir tiempos, distancias y consumo de energia. El planteamiento propuesto está inspirado en el enfoque de asignación de tareas basado en subasta, analizado en varias de sus vertientes en (Schneider et al., 2014; Schneider et al., 2015).

Para conferir una mayor verosimilitud y transferencia del proyecto a la aplicación real AGRICOBIOT, se ha supuesto que las tareas que debe desempeñar este sistema son fundamentalmente de transporte. Por ello, una visión lógica pasaría por hacer corresponder a estas tareas con una acción de transporte, es decir, cubrir un trayecto desde un punto inicial de recogida hasta un punto final de entrega. De esta manera, todos los encargos de transporte deben ser realizados de forma que se minimice el tiempo invertido en completarlo y a su vez que los robots estén activos el mayor tiempo posible.

De todas las alternativas, se ha implementado la asignación de tareas basada en subasta ordenada de elementos individuales (Schneider et al., 2015). En primer lugar, el subastador central es el encargado de repartir las tareas, y publica la primera de ellas, tras lo cual se permite que los robots pujen durante un breve período de tiempo. La puja de cada uno de ellos es, en este caso, el tiempo estimado en llegar al punto de recogida más el tiempo del encargo de transporte. Este valor estará influenciado por la posición en la que se encuentra el robot en el momento de la puja, la trayectoria planificada por su nodo de navegación y otros parámetros como la velocidad máxima de desplazamiento. Finalizada la subasta, se anuncia el agente ”ganador”, que procede a completar el encargo. En la misma línea se subastan las siguientes tareas hasta que todos los robots se encuentran activos.

Capítulo 4

Resultados

Diseñar y construir un sistema compuesto por múltiples robots autónomos que trabajen de forma cooperativa no es un asunto trivial. Si además se debe realizar una distribución eficiente de tareas y tener la capacidad para esquivarse entre ellos y evitar obstáculos estáticos y dinámicos del entorno, la solución puede tornarse algo compleja. En este capítulo se explica de manera pormenorizada todos los mecanismos, algoritmos y componentes empleados para tal fin.

En primer lugar, se debe conocer qué robots se van a emplear como base para este cometido y cuáles son sus características. Esta parte se expone en el apartado 4.1. Para controlar estas máquinas es fundamental elegir una buena arquitectura de control y un software adecuado que permita interactuar con ellos, lo cual se describe en el apartado 4.2. Una vez están instaladas todas las herramientas se comienza a trabajar en simulación, para lo cual es recomendable tener un modelo lo más fiel posible al robot real, como se construye en el apartado 4.3.

A continuación, se desmiembra el problema general en partes más asequibles y manejables. La primera de ellas es la representación del entorno en un mapa para su posterior uso en navegación, como se lleva a cabo en el apartado 4.4. Posteriormente, se desarrollan los algoritmos de navegación autónoma que permitan al robot desplazarse por el mundo, realizado en el apartado 4.6. Un paso más allá es el que se muestra en el apartado 4.8, donde se programa para poder evitar obstáculos y poder así navegar múltiples robots en un espacio compartido y sin colisiones. Por último, se propone un mecanismo para distribuir de forma equilibrada un conjunto de tareas entre todos ellos en el apartado 4.10, obteniendo finalmente un sistema multi-robot totalmente autónomo. De forma paralela, se realiza la implementación de estos métodos en los robots reales, lo cual se demuestra en los apartados 4.5, 4.7, 4.9 y 4.11, respectivamente.

4.1. Arquitectura hardware del prototipo final

La relación de conectividad entre todos los periféricos a bordo de los robots móviles se refleja en la figura 4.1.1.

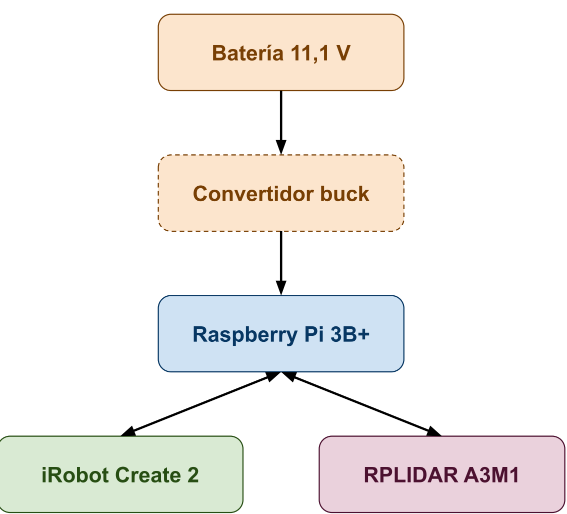


Figura 4.1.1: Arquitectura hardware

Con todos los componentes detallados en los apartados anteriores, el prototipo final con el que se han desarrollado e implementado los algoritmos de navegación autónoma se muestra en la figura 4.1.2.

Figura 4.1.2: Prototipo final

Como ya se contempló en el apartado 3.1.3, la ventana óptica del lidar debe quedar por encima del resto de periféricos una vez estén todos colocados sobre la plataforma superior del robot, como se observa en la figura 4.1.3.

Figura 4.1.3: Prototipo final (vista frontal)

4.2. Arquitectura software propuesta

Llevar a cabo la localización, navegación y coordinación simultánea de una flota de robots es una tarea que requiere una infraestructura que permita tanto procesar los datos recibidos por los sensores y adecuar la respuesta óptima como una comunicación estable entre los distintos nodos de computación. La arquitectura hardware-software que se ha empleado, de forma ilustrativa y a grandes rasgos, es la mostrada en la figura 4.2.1. Cabe destacar en este punto que las pruebas finales con ambos robots se han tenido que realizar con otro ordenador de a bordo en uno de ellos, debido a las restricciones de acceso a los laboratorios como consecuencia del estado de alarma.

Como núcleo central de procesamiento y distribución de tareas se encuentra la unidad central, el nodo más potente, que corresponde a un ordenador portátil externo. En el caso de las unidades móviles, se ha optado por una Raspberry Pi 3B+ para establecer la comunicación con el Create 2.

El empleo de un nodo central se debe fundamentalmente a la falta de capacidad de cómputo de las unidades móviles en la implementación real. Fuera del ámbito académico en el que se enmarca este trabajo, y teniendo en mente una posible aplicación comercial, se instalarían equipos de mayor potencia en los robots móviles. Por esta razón, los algoritmos de navegación se han diseñado desde un enfoque descentralizado, por lo que este nodo sería innecesario. Igualmente, en este nodo se han llevado a cabo las simulaciones del sistema previas a la puesta en funcionamiento real.

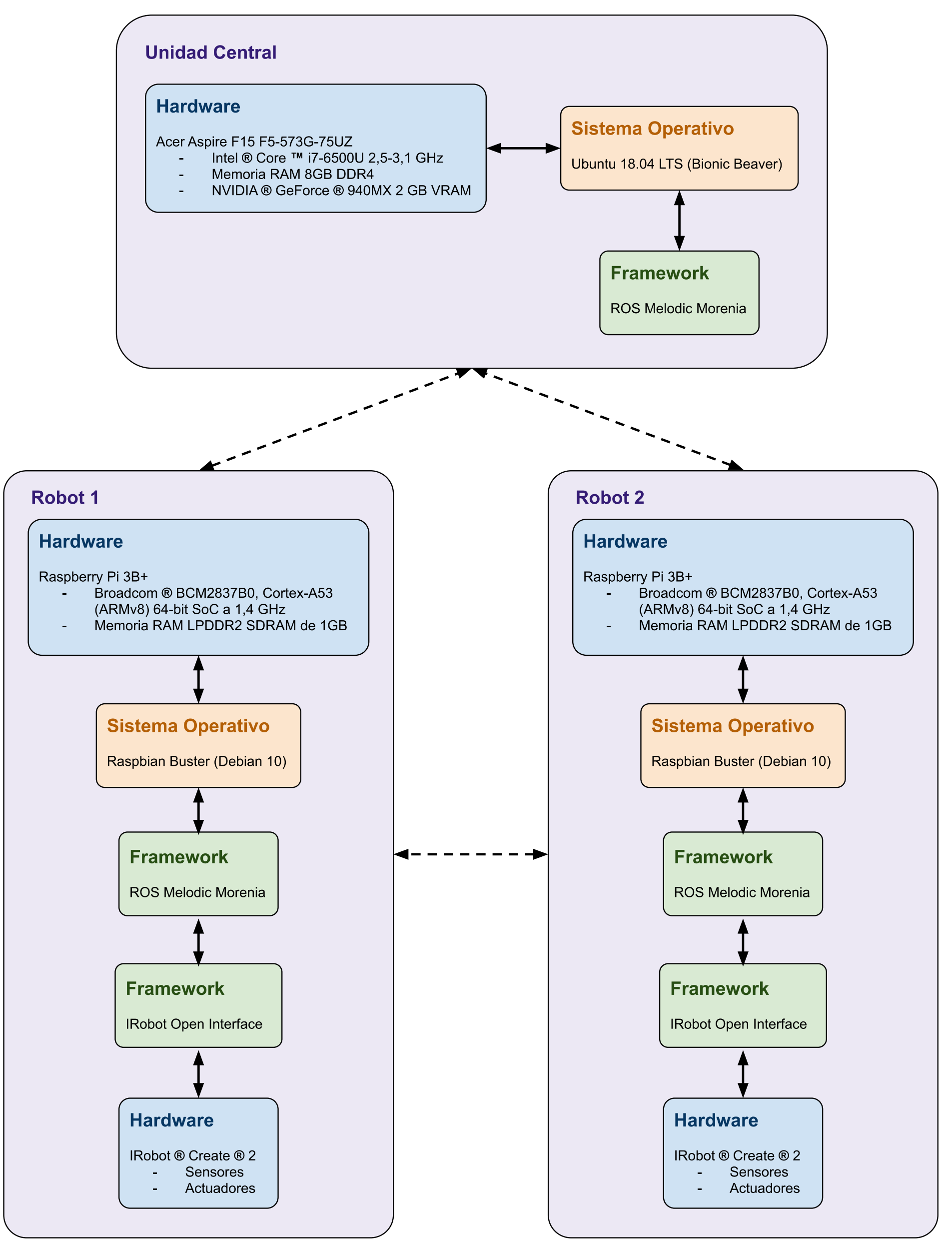


Figura 4.2.1: Arquitectura software

4.3. Modelado del robot en ROS y Gazebo

Los modelos URDF que proporciona el paquete ca_description no son lo suficientemente exactos, ya que la configuración de los controladores del robot diferencial tienen las ruedas cambiadas de orden, por lo que el robot no sigue los comandos de velocidad adecuadamente. Además, al añadir al árbol de *links* en URDF el modelo correspondiente al lidar, éste no queda bien orientado, ya que se inclina con un cierto ángulo hacia el suelo, devolviendo algunas mediciones erróneas (figura 4.3.1).

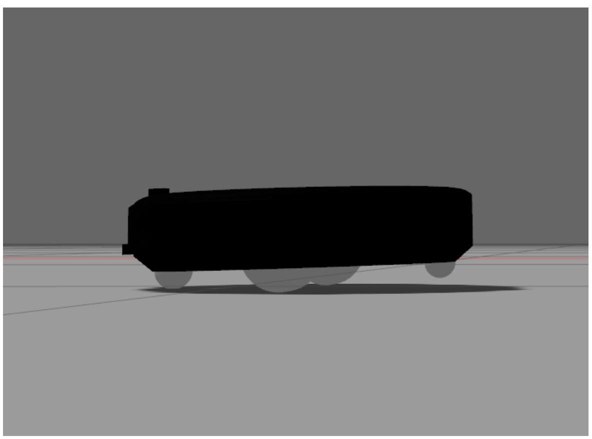


Figura 4.3.1: Errores del modelo previo

Por ello, y con objetivo de reproducir finalmente un modelo robusto y fiel al robot Create 2 sin estos pequeños errores, se ha modelado el robot desde cero, siguiendo las medidas proporcionadas por el fabricante (iRobot, 2018).

Como ya se vió en el apartado 3.2.3, el formato de los modelos diseñados para ROS son algo distintos a los de Gazebo. Para solventar este problema, se han creado creado dos archivos: el primero de ellos (.xacro) donde se incorpora todo el árbol de enlaces para ROS y un segundo (.gazebo) con las etiquetas necesarias para su compatibilidad con Gazebo.

Para mejorar la legibilidad del código, se han empleado las macros de Xacro (Open Robotics, 2020g). Con ellas, se crean piezas de código parametrizadas y reutilizables, como los bloques de información inercial de los enlaces del robot. Se comienza definiendo una serie de parámetros: el diámetro de la base y las ruedas, separación entre ellas y grosor de la base, entre otros.

```
1 <?xml version="1.0"?>
2 <!-- iRobot Roomba 650 model -->
```

```
3 <robot name="roomba" xmlns:xacro="http://www.ros.org/wiki/xacro">
4
5 <!-- Robot form parameters -->
6   <xacro:property name="PI" value="3.141592"/>
7   <xacro:property name="wheels_offset" value="0.235"/>
8   <xacro:property name="wheels_xgap" value="0.0"/>
9   <xacro:property name="wheels_radius" value="0.036"/>
10  <xacro:property name="wheels_width" value="0.015"/>
11  <xacro:property name="wheels_mass" value="0.7"/>
12  <xacro:property name="wheels_effort" value="1"/>
13  <xacro:property name="wheels_velocity" value="1"/>
14  <xacro:property name="wheels_damping" value="0.1"/>
15  <xacro:property name="base_zgap" value="0.01"/>
16  <xacro:property name="base_height" value="${0.083-base_zgap}"/>
17  <xacro:property name="base_radius" value="0.174"/>
18  <xacro:property name="base_mass" value="3.0"/>
19  <xacro:property name="roller_radius" value="${base_zgap*0.9}"/>
20  <xacro:property name="roller_xgap" value="0.135"/>
21  <xacro:property name="ir_height" value="0.009"/>
22  <xacro:property name="ir_radius" value="0.02"/>
23  <xacro:property name="ir_mass" value="0.01"/>
24  <xacro:property name="ir_xgap" value="0.15"/>
25  <xacro:property name="lidar_radius" value="0.038"/>
26  <xacro:property name="lidar_height" value="0.041"/>
27  <xacro:property name="lidar_offset_y" value="0.105"/>
```

create2.xacro

La definición de la macro correspondiende a la inercia de un sólido cilíndrico es la siguiente:

```
28  <xacro:macro name="cylinder_inertial" params="radius length mass">
29    <inertial>
30      <origin xyz="0 0 0" rpy="0 0 0"/>
31      <mass value="${mass}"/>
32      <inertia ixx="${mass*radius*radius/2}" ixy="0.0" ixz="0.0"
33              iyy="${mass*radius*radius/4+mass*length*length/12}"
                   iyz="0.0"
34              izz="${mass*radius*radius/4+mass*length*length/12}"/>
35    </inertial>
```

```
36 </xacro:macro>
```

create2.xacro

Posteriormente, se deben incluir los archivos de las etiquetas necesarias en Gazebo y las referencias a los materiales y colores de cada enlace (también se podrían incluir al final).

```
37 <xacro:include filename="$(find tfg_simulations)/urdf/create2.gazebo"
       />
38 <xacro:include filename="$(find tfg_simulations)/urdf/materials.xacro"
       />
```

create2.xacro

Ahora sí, se inicia con la definición de los primeros enlaces del robot. La estructura que sigue este tipo de modelos es una estructura de árbol, en el cual se crean una serie de enlaces *links* unidos entre sí mediante articulaciones *joints*. El primero de ellos será la huella de nuestra base (base_footprint). Corresponde a la proyección ortogonal del enlace principal (base_link) sobre el suelo. No es absolutamente necesario, pero se recomienda su uso para una representación más estable (Moulard, 2011). De forma empírica, se ha comprobado que realizar el mapeo respecto a este enlace proyecta el mapa a la misma altura, es decir, en el suelo, lo cual evita solapamientos visuales con otras entidades.

```
39 <link name="base_footprint">
40   <inertial>
41     <origin xyz="0 0 0" rpy="0 0 0"/>
42     <mass value="0.01"/>
43     <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
44              iyy="0.0001" iyz="0.0" izz="0.0001"/>
45   </inertial>
46   <visual>
47     <origin xyz="0 0 0" rpy="0 0 0"/>
48     <geometry>
49       <mesh
           filename="package://tfg_simulations/meshes/base_create2.dae"
           />
50     </geometry>
51   </visual>
```

```
52      <collision>
53        <origin xyz="0 0 0.05" rpy="0 0 0"/>
54        <geometry>
55          <box size="0.001 0.001 0.001" />
56        </geometry>
57      </collision>
58    </link>
59
60    <joint name="base_footprint_joint" type="fixed">
61      <origin xyz="0 0 ${(base_height/2+base_zgap)}" rpy="0 0 0"/>
62      <parent link="base_footprint"/>
63      <child link="base_link"/>
64    </joint>
65
66    <link name="base_link">
67      <xacro:cylinder_inertial
68        radius="${base_radius}" length="${base_height}"
            mass="${base_mass}"/>
69      <collision>
70        <origin xyz="0 0 0" rpy="0 0 0"/>
71        <geometry>
72          <cylinder radius="${base_radius}" length="${base_height}"/>
73        </geometry>
74      </collision>
75    </link>
```

create2.xacro

Para definir completamente en enlace y que en motor de físicas de Gazebo sea capaz de simularlo de forma realista, se tienen que cumplimentar las etiquetas *inertial*, *collision* y *visual*. La primera lo define en cuanto a sus momentos de inercia, es decir, su masa y cómo se distribuye en su geometría. La etiqueta *collision* lo parametriza para su interacción con el entorno y otros enlaces. Por último, el aspecto visual no es más que la apariencia gráfica y no es imprescindible. Por esta razón, se importa la malla del robot (aproximación poligonal) en el apartado visual (Mechatronics Art, 2020), ya que si se hiciera en el apartado de colisiones, el motor de físicas tardaría más en resolver el modelo físico. De esta forma se consigue un sólido tridimensional de menor resolución para la resolución en Gazebo mientras que se conserva un aspecto visual realista.

De igual manera, se procede con el enlace del lidar:

```
76  <joint name="lidar_joint" type="fixed">
77    <origin xyz="0 ${lidar_offset_y} ${base_height/2+lidar_height/2}"
78            rpy="0 0 0"/>
79    <parent link="base_link"/>
80    <child link="lidar"/>
81  </joint>
82
83  <link name="lidar">
84    <inertial>
85      <origin xyz="0 0 0" rpy="0 0 0"/>
86      <mass value="0.01"/>
87      <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
88              iyy="0.0001" iyz="0.0" izz="0.0001"/>
89    </inertial>
90    <visual>
91      <origin xyz="0 0 ${-lidar_height/2}" rpy="0 0 0"/>
92      <geometry>
93        <mesh
            filename="package://tfg_simulations/meshes/rplidar_a3m1.dae"
            />
94      </geometry>
95      <material name="purple"/>
96    </visual>
97    <collision>
98      <origin xyz="0 0 0" rpy="0 0 0"/>
99      <geometry>
100       <cylinder radius="${lidar_radius}" length="${lidar_height}"/>
101     </geometry>
102   </collision>
103 </link>
```

create2.xacro

El resto del modelo seguiría con la misma estructura, cambiando el tipo de articulación *fixed* por *continuous* en el caso de las ruedas. Se definen, además, los enlaces roller para modelar la rueda delantera de apoyo e ir para aproximar el peso que tiene el robot real en la parte delantera y que evita que se levante en exceso al acelerar. El resto del código se incluye en el anexo.

Paralelamente, el archivo con las etiquetas de Gazebo debe contener una referencia a cada enlace del árbol en URDF. Un ejemplo con los principales links es el siguiente:

```
1  <gazebo reference="base_footprint">
2    <mu1>0</mu1>
3    <mu2>0</mu2>
4  </gazebo>
5
6  <gazebo reference="base_link">
7    <mu1>0</mu1>
8    <mu2>0</mu2>
9  </gazebo>
10
11 <gazebo reference="right_wheel">
12   <mu1>0.8</mu1>
13   <mu2>0.8</mu2>
14   <material>Gazebo/Black</material>
15 </gazebo>
16
17 <gazebo reference="left_wheel">
18   <mu1>0.8</mu1>
19   <mu2>0.8</mu2>
20   <material>Gazebo/Black</material>
21 </gazebo>
```

create2.gazebo

Para indicarle a Gazebo que uno de esos enlaces (lidar) es un dispositivo tipo escáner láser, se utiliza el correspondiente *plugin*. Los plugins en Gazebo son herramientas predefinidas que simulan el comportamiento de los dispositivos sensores más empleados en robótica, como láseres, cámaras o IMUs, entre otros (Open Robotics, 2020b). Como parámetros, se indican el rango máximo y mínimo de medición o la frecuencia de muestreo, según el fabricante (Slamtec, 2019). Se añade algo de ruido para tener un comportamiento más realista y se indican los tópicos en los que se quiere que publique la información y en referencia a qué enlace, en este caso, a *base_link*.

```
22 <gazebo reference="lidar">
23   <mu1>0.5</mu1>
24   <mu2>0.5</mu2>
```

```
25    <material>Gazebo/Purple</material>
26    <sensor type="ray" name="laser_base">
27      <pose>0 0 0 0 0 0</pose>
28      <visualize>false</visualize>
29      <update_rate>20</update_rate>
30      <ray>
31        <scan>
32          <horizontal>
33            <samples>720</samples>
34            <resolution>1</resolution>
35            <min_angle>-3.141592</min_angle>
36            <max_angle>3.141592</max_angle>
37          </horizontal>
38        </scan>
39        <range>
40          <min>0.10</min>
41          <max>25.0</max>
42          <resolution>0.01</resolution>
43        </range>
44        <noise>
45          <type>gaussian</type>
46          <!-- Noise parameters based on published spec for Hokuyo laser
47               achieving "+-30mm" accuracy at range < 10m. A mean of 0.0m
                     and
48               stddev of 0.01m will put 99.7% of samples within 0.03m of
                     the true
49               reading. -->
50          <mean>0.0</mean>
51          <stddev>0.01</stddev>
52        </noise>
53      </ray>
54      <plugin name="gazebo_ros_head_hokuyo_controller"
          filename="libgazebo_ros_laser.so">
55        <topicName>scan</topicName>
56        <frameName>base_footprint</frameName>
57      </plugin>
58    </sensor>
59  </gazebo>
```

create2.gazebo

Otro plugin necesario es el que permite implementar la interfaz entre los comandos de velocidad del robot y las órdenes de esfuerzo a las ruedas: el controlador de robots diferenciales.

```
60  <gazebo>
61    <plugin name="differential_drive_controller"
        filename="libgazebo_ros_diff_drive.so">
62      <alwaysOn>true</alwaysOn>
63      <updateRate>50</updateRate>
64      <leftJoint>left_wheel_joint</leftJoint>
65      <rightJoint>right_wheel_joint</rightJoint>
66      <wheelSeparation>0.235</wheelSeparation>
67      <wheelDiameter>0.072</wheelDiameter>
68      <wheelTorque>20</wheelTorque>
69      <wheelAcceleration>5</wheelAcceleration>
70      <commandTopic>cmd_vel</commandTopic>
71      <odometryTopic>odom</odometryTopic>
72      <odometryFrame>odom</odometryFrame>
73      <robotBaseFrame>base_footprint</robotBaseFrame>
74      <publishOdomTF>true</publishOdomTF>
75    </plugin>
76  </gazebo>
```

create2.gazebo

Con esto, se concluye el modelo completo del robot Create 2 (figura 4.3.2).

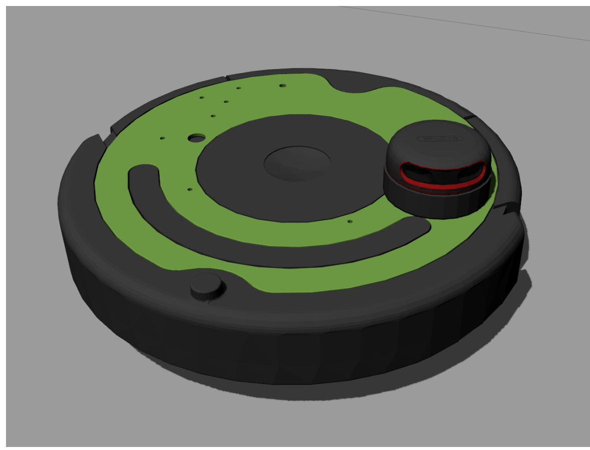


Figura 4.3.2: Modelo final

También se ha compuesto un segundo modelo con un aspecto visual más simple, para conseguir una renderización más rápida en el caso de simularlo con un equipo de bajas prestaciones (create2_basic.xacro). Esto sirve para mostrar una comparativa entre la estética visual del robot y la geometría real que se usa en el cómputo (figura 4.3.3).

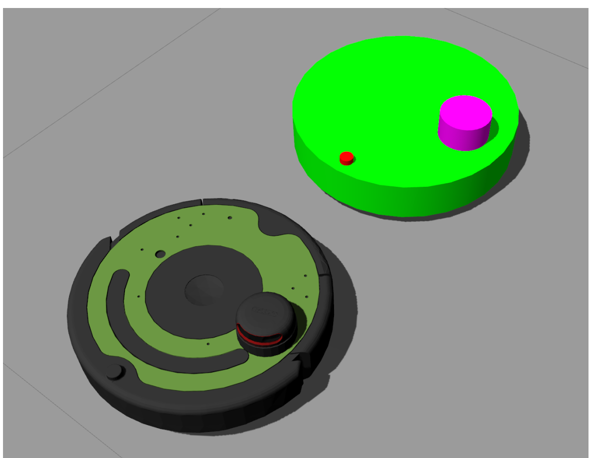


Figura 4.3.3: Comparativa de modelos (visual y de colisiones)

Para mover el robot en el mundo virtual (y más adelante en el real) se ha confeccionado un programa para enviar comandos de velocidad al robot desde un mando tipo joystick Dualshock®3 (Sony Corporation, 2020). Para ello, se ha hecho uso de la biblioteca pygame para Python 3 (Pygame, 2020).

Con finalidad de conseguir una mayor modularidad, se crean dos nodos para esta función. El primero de ellos (ps3_node.py) lee las entradas que se dan al mando y las publica en tópico *axis*, con la posibilidad de aumentar el número de ejes y botones publicados ya que se trata de un mensaje propio (Joystick.msg).

```
1 #!/usr/bin/env python3
2 import rospy
3 import tfg_simulations.msg
4 import pygame
5
6 pygame.init()
7 pygame.joystick.init()
8 joystick = pygame.joystick.Joystick(0)
9 joystick.init()
10
11 print ('Initialized Joystick : %s' % joystick.get_name())
12 # pygame.display.set_mode((200, 200))
13
```

```
14 if __name__ == '__main__':
15     rospy.init_node('joystick_controller')
16     pub = rospy.Publisher('axis', tfg_simulations.msg.Joystick,
           queue_size=1)
17     rate = rospy.Rate(50.0)
18 
19     numaxes = joystick.get_numaxes()
20     print("numaxes")
21     print(numaxes)
22     print("--------------")
23     numbuttons = joystick.get_numbuttons()
24     print("numbuttons")
25     print(numbuttons)
26     print("--------------")
27 
28     while not rospy.is_shutdown():
29         cmd = tfg_simulations.msg.Joystick()
30         pygame.event.pump()
31         cmd.l3x = joystick.get_axis(0)
32         cmd.l3y = joystick.get_axis(1)
33         cmd.r3x = joystick.get_axis(3)
34         cmd.r3y = joystick.get_axis(4)
35 
36         pub.publish(cmd)
37         rate.sleep()
```

ps3_node.py

El segundo (create_cmd.py) se encarga de publicar el correspondiente comando de velocidad.

```
1 #!/usr/bin/env python3
2 import rospy
3 import tfg_simulations.msg
4 from geometry_msgs.msg import Twist
5 
6 vel_max = 0.5 # Max Create linear velocity
7 w_max = 4.25 # Max Create angular velocity
8 vel = 0
9 w = 0
```

```
10
11 def callback(msg):
12     global vel, w
13     vel = msg.r3y * (-1) * vel_max
14     w = msg.l3x * (-1) * w_max
15
16
17 if __name__ == '__main__':
18     rospy.init_node('create_cmd')
19     pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
20     rate = rospy.Rate(10.0)
21
22     cmd = Twist()
23     cmd.linear.x = 0
24     cmd.linear.y = 0
25     cmd.linear.z = 0
26     cmd.angular.x = 0
27     cmd.angular.y = 0
28     cmd.angular.z = 0
29
30     rospy.Subscriber('axis', tfg_simulations.msg.Joystick, callback)
31
32     while not rospy.is_shutdown():
33         cmd.linear.x = vel
34         cmd.angular.z = w
35         pub.publish(cmd)
36         rate.sleep()
```

create_cmd.py

Se ha comprobado empíricamente que la máxima velocidad lineal del create 2 es de 0,5 m/s y su velocidad angular de 4,25 rad/s. También se ha creado un tercer programa con el mismo propósito pero con el teclado (keyboard_pub.py).

4.4. Mapeo y localización simultáneos (SLAM) en simulación

Como ya se describió en apartado 2.1, el lidar va a permitir realizar un mapeo del entorno. Este mapa será empleado posteriormente para la localización y navegación autónoma. Para ello, se requiere iniciar una serie de nodos. En primer lugar, se indica

a Gazebo que se inicialice con la interfaz gráfica y que cargue el mundo creado para la simulación.

```
1 <launch>
2
3   <!-- Parameters configuration -->
4   <arg name="paused" default="false"/>
5   <arg name="use_sim_time" default="true"/>
6   <arg name="gui" default="true"/>
7   <arg name="headless" default="false"/>
8   <arg name="debug" default="false"/>
9
10  <!--Gazebo parameters -->
11  <include file="$(find gazebo_ros)/launch/empty_world.launch">
12    <arg name="world_name" value="$(find
        tfg_simulations)/worlds/p1_slam.world"/>
13    <arg name="debug" value="$(arg debug)" />
14    <arg name="gui" value="$(arg gui)" />
15    <arg name="paused" value="$(arg paused)"/>
16    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
17    <arg name="headless" value="$(arg headless)"/>
18  </include>
```

p1_slam.launch

A continuación, se carga en el servidor de parámetros la descripción del robot creada en el apartado 4.3.

```
20  <param name="robot_description"
21    command="$(find xacro)/xacro '$(find
        tfg_simulations)/urdf/create2.xacro'" />
```

p1_slam.launch

Ese modelo cargado en el servidor se hace aparecer en Gazebo.

```
22  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
      respawn="false" output="screen"
23    args="-urdf -model create -param robot_description"/>
```

p1_slam.launch

Se añade el nodo *joint_state_publisher*, sin interfaz gráfica, ya que los movimientos seran controlados mediante un *joystick* y con una tasa de actualización de 20 hz. Este nodo se encargará de publicar el estado de las articulaciones móviles del robot, en este caso, las ruedas.

```
24  <node name="joint_state_publisher" pkg="joint_state_publisher"
        type="joint_state_publisher">
25    <param name="use_gui" value="false"/>
26    <param name="rate" value="20"/>
27  </node>
```

p1_slam.launch

Posteriormente, el nodo *robot_state_publisher* llevará a cabo la publicación de las transformadas (matrices de traslación y rotación) entre los elementos del robot

```
28  <node name="robot_state_publisher" pkg="robot_state_publisher"
        type="robot_state_publisher">
29    <param name="publish_frequency" value="60.0"/>
30    <param name="use_tf_static" value="true"/>
31  </node>
```

p1_slam.launch

Con toda esta información, el nodo mapping es el que va realizando un seguimiento del recorrido del robot y la información que recibe del entorno a través del lidar, construyendo su correspondiente mapa.

```
32  <node name="mapping" pkg="gmapping" type="slam_gmapping"
        args="scan:=scan">
33    <param name="base_frame" value="base_link"/>
34    <param name="map_update_interval" value="0.5"/>
35  </node>
```

p1_slam.launch

A continuación, se lanzan dos nodos propios del paquete *tfg_simulations*. Se han creado dos diferenciados para conferir una mayor modularidad al proyecto. El primero de ellos recoge los estados de los *joysticks* del mando para su publicación,

mientras que el segundo escucha este mensaje y lo transforma a un comando de velocidad para el robot. Se detallarán a continuación.

```
36   <node name="joystick_publisher" pkg="tfg_simulations"
         type="ps3_node.py" respawn="true" />
37   <node name="joystick_to_cmd" pkg="tfg_simulations"
         type="create_cmd.py"/>
```

p1_slam.launch

Por último, se visualiza todo en RVIZ, cargando su correspondiente archivo de configuración.

```
38   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
         tfg_simulations)/config/p1_slam.rviz"
39     respawn="true"/>
40
41 </launch>
```

p1_slam.launch

Por último, se incluyen los nodos de los comandos de velocidad del mando y el nodo de mapeo mediande SLAM. El nodo final corresponde a RVIZ, la herramienta de visualización de ROS.

Como se comentaba anteriormente, los nodos para dirigir el robot son los siguientes. En *ps3_node*, se importan los paquetes y definiciones de mensajes y se configura la comunicación con la biblioteca *pygame* que proporciona una interfaz directa con el mando.

```
38 #!/usr/bin/env python3
39 import rospy
40 import tfg_simulations.msg
41 import pygame
42
43 pygame.init()
44 pygame.joystick.init()
45 joystick = pygame.joystick.Joystick(0)
46 joystick.init()
```

ps3_node.py

Posteriormente, se inicia el nodo *joystick_controller* que publica en el tópico ”axis” con una frecuencia de 50 hz.

```
47 if __name__ == '__main__':
48     rospy.init_node('joystick_controller')
49     pub = rospy.Publisher('axis', tfg_simulations.msg.Joystick,
           queue_size=1)
50     rate = rospy.Rate(50.0)
```

ps3_node.py

Por último, el bucle consiste en leer el estado de los *joysticks* y publicarlo.

```
51 while not rospy.is_shutdown():
52         cmd = tfg_simulations.msg.Joystick()
53         pygame.event.pump()
54         cmd.l3x = joystick.get_axis(0)
55         cmd.l3y = joystick.get_axis(1)
56         cmd.r3x = joystick.get_axis(3)
57         cmd.r3y = joystick.get_axis(4)
58
59         pub.publish(cmd)
60         rate.sleep()
```

ps3_node.py

En segundo nodo es el *create_cmd*, que se encarga de enviar el comando de velocidad. En la misma línea, se importan las dependencias al inicio. También se parametrizan las velocidades máximas del robot real.

```
37 #!/usr/bin/env python3
38 import rospy
39 import tfg_simulations.msg
40 from geometry_msgs.msg import Twist
41
42 vel_max = 0.5 # Max Create linear velocity
43 w_max = 4.25 # Max Create angular velocity
44 vel = 0
45 w = 0
```

create_cmd.py

Se define una función de devolución de llamada (*callback*) para procesar el mensaje que se reciba en el tópico al que se suscribe.

```
46 def callback(msg):
47     global vel, w
48     vel = msg.r3y * (-1) * vel_max
49     w = msg.l3x * (-1) * w_max
```

create_cmd.py

Igual que en caso anterior, se configuran los nombres de nodos y tópicos de publicación y suscripción.

```
50 if __name__ == '__main__':
51     rospy.init_node('create_cmd')
52     pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
53     rate = rospy.Rate(25.0)
54     rospy.Subscriber('axis', tfg_simulations.msg.Joystick, callback)
55     cmd = Twist()
```

create_cmd.py

Y con un sencillo bucle se publica el comando.

```
56     while not rospy.is_shutdown():
57             cmd.linear.x = vel
58             cmd.angular.z = w
59             pub.publish(cmd)
60             rate.sleep()
```

create_cmd.py

Las definiciones de mensajes, así como todo el código empleado se incluyen en el anexo del documento. En la misma línea, se ha creado otro *script* para realizar la misma acción mediante teclado del ordenador (*keyboard_pub.py*).

De esta forma, controlando el robot mediante el mando, se recorre todo el entorno hasta tener el mapa completo (figura 4.4.1).

Se compara y se valida el mapa creado con el mapa real (figura 4.4.2).

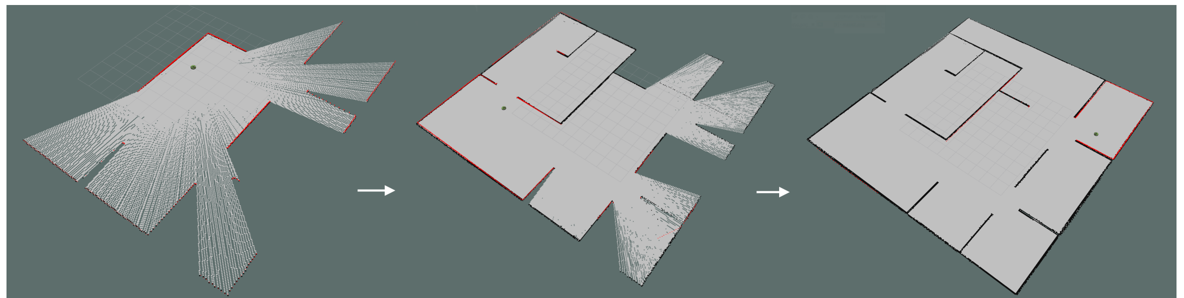


Figura 4.4.1: Proceso de mapeo

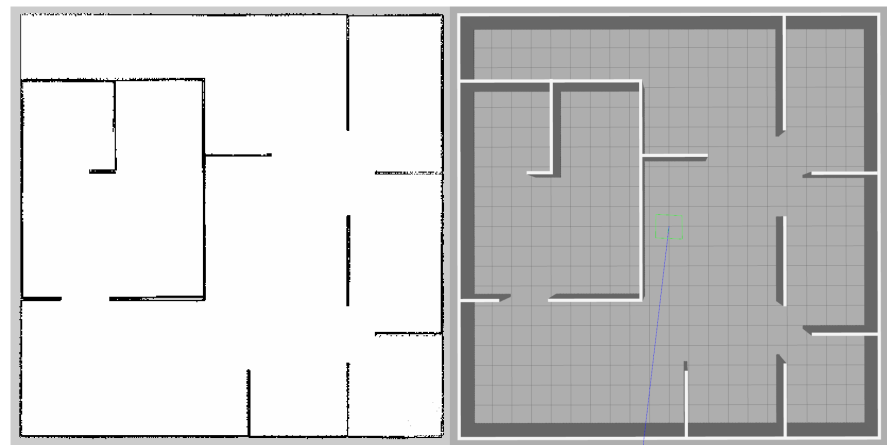


Figura 4.4.2: Mapa construido (izquierda) y mapa real (derecha)

4.5. Mapeo y localización simultáneos (SLAM) real

Para realizar un mapeo del entorno real, los pasos a seguir son similares a los descritos en el anterior apartado. Sin embargo, en este caso la información no se va a obtener del simulador, sino del robot real. Para ello, se establece la comunicación con él vía SSH para lanzar el archivo *.launch* que contiene todos los nodos que permiten la interfaz con los actuadores y con el lidar.

En primer lugar, se lanza el nodo *ca_driver*, indicando el puerto de comunicación, USB1 en este caso y algunos parámetros de configuración.

```
42 <?xml version="1.0"?>
43 <launch>
44
45   <arg name="config" default="$(find tfg_remote)/config/multi.yaml" />
46   <arg name="desc" default="true" />
47
```

```
48  <node name="ca_driver" pkg="ca_driver" type="ca_driver"
      output="screen">
49    <rosparam command="load" file="$(arg config)" />
50    <param name="robot_model" value="CREATE_2" />
51    <param name="dev" value="/dev/ttyUSB1" />
52  </node>
```

p1_slam.launch

Al igual que anteriormente, se ejecutan el nodo *robot_state_publisher* y se publica el modelo del robot en el servidor de parámetros. Para navegar con el robot real, se ha decidido usar el modelo por defecto del robot que viene en *ca_description* y publicar una transformada estática del marco de referencia del lidar. El propósito de este cambio es agilizar la configuración de la posición del lidar, ya que en uno de los robots será central y en otro estará algo desplazada hacia un lado. De esta forma con modificar numéricamente los tres primeros argumentos del nodo *tf2_ros* se puede modificar su posición.

```
53 <param name="robot_description" command="$(find xacro)/xacro.py '$(find
     ca_description)/urdf/create_2.urdf.xacro'" />
54
55  <node name="robot_state_publisher" pkg="robot_state_publisher"
      type="robot_state_publisher" output="screen" >
56    <param name="tf_prefix" value="create_1"/>
57  </node>
58
59  <node name="rplidarNode" pkg="rplidar_ros" type="rplidarNode"
      output="screen" args="scan:=/create_1/scan">
60    <param name="serial_port" type="string" value="/dev/ttyUSB0"/>
61    <param name="serial_baudrate" type="int" value="256000"/><!--A3 -->
62    <param name="frame_id" type="string" value="create_1/lidar"/>
63    <param name="inverted" type="bool" value="false"/>
64    <param name="angle_compensate" type="bool" value="true"/>
65    <param name="scan_mode" type="string" value="Sensitivity"/>
66  </node>
67
68  <node pkg="tf2_ros" type="static_transform_publisher" name="lidar_tf"
      args="-0.01 0.105 0.055 3.041592 0 0 create_1/base_link
      create_1/lidar" />
69
```

```
70 </launch>
```

p1_slam.launch

Esta será la configuración por defecto del robot en la mayoría de las ocasiones, variando ligeramente en el caso multi-robot por cuestiones de espacios de trabajo (*namespaces*). El resto de nodos se han ejecutado en el ordenador personal, debido a su mayor potencia.

La navegación se realiza de forma teledirigida mediante mando y se reduce la velocidad del robot deliberadamente para conseguir una mayor precisión.

Finalmente, se obtiene el mapa final de todo el entorno (figura 4.5.1).

Figura 4.5.1: Mapa real

Se puede observar en la parte superior derecha cómo el lidar tiene una rendimiento ostensiblemente inferior y no consigue mapear bien los límites de la habitación. En esta zona se encuentran ventanas y muebles con cristales. Esto provoca que los haces láser no se reflejen de la misma forma que en objetos opacos, o que incluso los atraviesen. Este inconveniente limita en ciertos casos el SLAM con los dispositivos lidar.

4.6. Navegación autónoma en simulación

Para poder asegurar una navegación autónoma, fiable y robusta, es necesario contar con un mapa del entorno, como el que se construyó en el apartado 4.4.

Posteriormente, el nodo de navegación autónoma construirá una trayectoria, si es posible, entre la posición del robot en cada momento y la posición destino.

En primer lugar, se lanza el nodo de servidor del mapa previamente construido.

```
1   <node pkg="map_server" type="map_server" args="$(find
       tfg_simulations)/map/map.yaml"
2     respawn="true" name="map_server" />
```

p2_autonav.launch

Además de los nodos descritos anteriormente, se carga el archivo que contiene los nodos restantes de localización y navegación. Se cargará un archivo u otro en función del planificador escogido.

```
3   <include file="$(find tfg_simulations)/launch/p2_move_base.launch" />
```

p2_autonav.launch

En este archivo se encuentran los nodos de localización mediante AMCL. AMCL realiza un seguimiento sobre la deriva que se produce en la referencia de odometría (*odom*) y la referencia del mapa (*map*). Los parámetros más importantes de su configuración son los tópicos de publicación para el escaneo y el número máximo y mínimo de partículas. También son importantes los parámetros de actualización *update_min_d* y *update_min_a* para indicar cada cuánto camino recorrido se debe reiniciar el filtro de partículas. Con *transform_tolerance* se indica cuánto tiempo hacia el futuro se publica la pose del robot y, por tanto, es válido su estado. Permite compensar retardos provocados por la comunicación, lo cual será fundamental en el despliegue con los Create 2. Como en la mayoría de casos, es necesario buscar el compromiso entre efectividad y coste computacional.

```
1 <launch>
2
3   <arg name="base_planner" default="false"/>
4   <arg name="teb_planner" default="false"/>
5
6 <!-- Publish scans from best pose at a max of 10 Hz -->
7   <node pkg="amcl" type="amcl" name="amcl" output="screen"
       args="scan:=scan map:=map">
8     <param name="use_map_topic" value="false"/>
9     <param name="base_frame_id" value="base_footprint"/>
```

```
10     <param name="odom_model_type" value="diff"/>
11     <param name="odom_alpha5" value="0.1"/>
12     <param name="transform_tolerance" value="0.1" />
13     <param name="gui_publish_rate" value="10.0"/>
14     <param name="laser_max_beams" value="30"/>
15     <param name="min_particles" value="500"/>
16     <param name="max_particles" value="5000"/>
17     <param name="kld_err" value="0.05"/>
18     <param name="kld_z" value="0.99"/>
19     <param name="odom_alpha1" value="0.2"/>
20     <param name="odom_alpha2" value="0.2"/>
21     <param name="odom_alpha3" value="0.8"/>
22     <param name="odom_alpha4" value="0.2"/>
23     <param name="laser_z_hit" value="0.5"/>
24     <param name="laser_z_short" value="0.05"/>
25     <param name="laser_z_max" value="0.05"/>
26     <param name="laser_z_rand" value="0.5"/>
27     <param name="laser_sigma_hit" value="0.2"/>
28     <param name="laser_lambda_short" value="0.1"/>
29     <param name="laser_lambda_short" value="0.1"/>
30     <param name="laser_model_type" value="likelihood_field"/>
31     <param name="laser_likelihood_max_dist" value="2.0"/>
32     <param name="update_min_d" value="0.2"/>
33     <param name="update_min_a" value="0.5"/>
34     <param name="odom_frame_id" value="odom"/>
35     <param name="resample_interval" value="1"/>
36     <param name="recovery_alpha_slow" value="0.0"/>
37     <param name="recovery_alpha_fast" value="0.0"/>
38   </node>
```

p2_move_base.launch

Para lanzar los algoritmos de navegación, basta con llamar al nodo *move_base* con los siguientes parámetros.

```
39   <node pkg="move_base" type="move_base" respawn="false"
         name="move_base_base" output="screen" >
40     <rosparam file="$(find
           tfg_simulations)/config/p2_base/costmap_common_params.yaml"
           command="load" ns="global_costmap" />
```

```
41      <rosparam file="$(find
          tfg_simulations)/config/p2_base/costmap_common_params.yaml"
          command="load" ns="local_costmap" />
42      <rosparam file="$(find
          tfg_simulations)/config/p2_base/local_costmap_params.yaml"
          command="load" />
43      <rosparam file="$(find
          tfg_simulations)/config/p2_base/global_costmap_params.yaml"
          command="load" />
44      <rosparam file="$(find
          tfg_simulations)/config/p2_base/planners_params.yaml"
          command="load" />
45
46      <param name="controller_frequency" value="10.0"/>
47      <param name="base_local_planner"
          value="base_local_planner/TrajectoryPlannerROS" />
48      <param name="base_global_planner"
          value="global_planner/GlobalPlanner" />
49    </node>
```

p2_move_base.launch

A su vez, cada uno de estos archivos a los que se llama, contienen los parámetros de configuración de los planificadores y los mapas de coste globales y locales. El primero de ellos contiene parámetros comunes a ambos mapas de coste, como los marcos de los sensores o cómo decae el coste de pasar cerca de las paredes en función de la distancia.

```
1 obstacle_range: 5.0
2 raytrace_range: 5.0
3 robot_radius: 0.174
4 cost_scaling_factor: 5.0 # How far from walls (more is further)
5 observation_sources: laser_scan_sensor
6 transform_tolerance: 0.1
7 laser_scan_sensor: {sensor_frame: lidar, data_type: LaserScan, topic:
    scan, marking: true, clearing: true}
```

costmap_common_params.launch

En el mapa de coste global se indican la frecuencia de actualización y publicación (si no fuera estático) con los datos del mapa local y el radio de inflado. Este radio de

inflado es el que define los costes y cómo aumentan o disminuyen estos con la distancia. De forma ilustrativa, en (Marder-Eppstein y Lu, 2020) se muestra cómo disminuye este coste (figura 4.6.1).

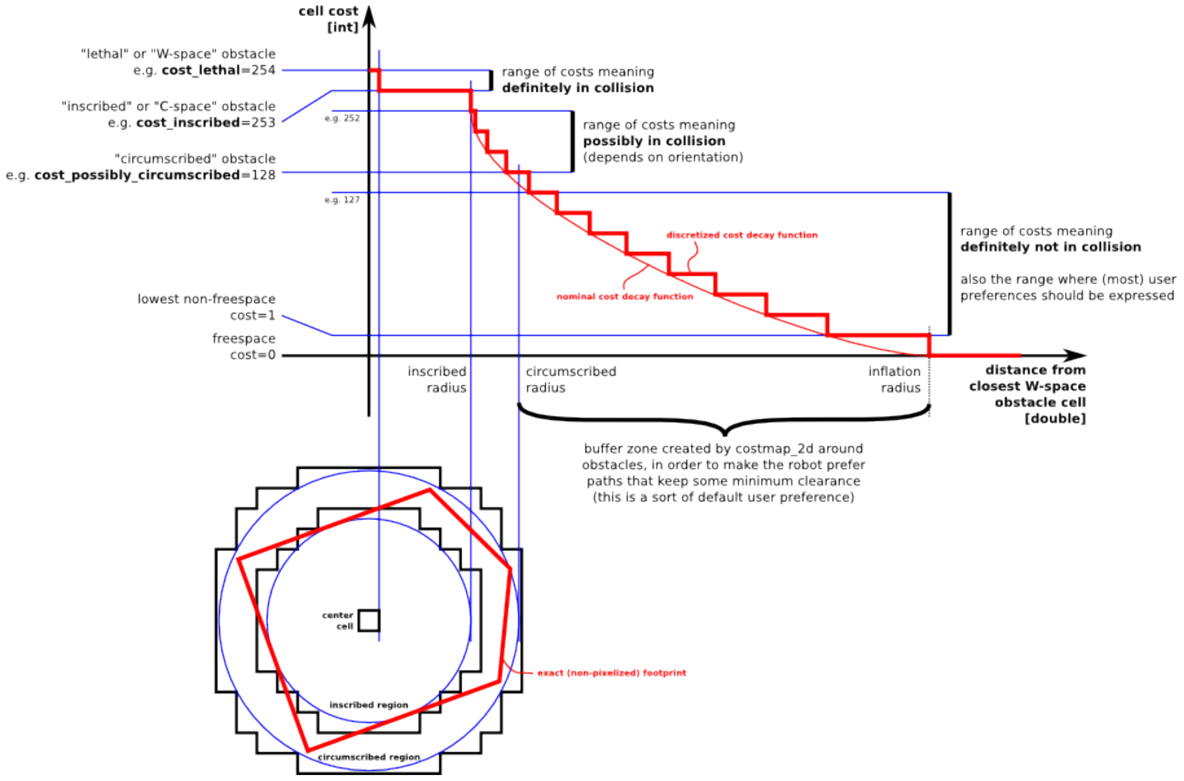


Figura 4.6.1: Coste según el radio de inflado

La pendiente de esta curva viene determinada por el parámetro *cost_scaling_factor*. En este caso, el mapa será estático.

```
1 global_costmap:
2   global_frame: odom
3   robot_base_frame: base_footprint
4   # update_frequency: 1.0
5   # publish_frequency: 1.0
6   static_map: true
7   inflation_radius: 0.5
8   resolution: 0.05
```

global_costmap_params.launch

De manera similar, el mapa de coste local incluye, adicionalmente, una ventana alrededor del robot sobre la que trabaja, para ahorrar coste computacional. Este

mapa sí será dinámico, ya que debe asegurarse de detectar posibles obstáculos, y debe actualizarse a una mayor frecuencia.

```
1 local_costmap:
2   global_frame: odom
3   robot_base_frame: base_footprint
4   update_frequency: 5.0
5   publish_frequency: 1.0
6   static_map: false
7   rolling_window: true
8   width: 3.0
9   height: 3.0
10  resolution: 0.05
11  inflation_radius: 0.4
```

local_costmap_params.launch

Por último se configuran los planificadores globales y locales. En el local, se indican los límites de velocidad y aceleración del robot, así como factores que ponderan cómo de próximo debe ser el plan local al global o cuánto debe separarse de los obstáculos. En el global, se indica la opción del algoritmo Dijkstra.

```
1 TrajectoryPlannerROS:
2   max_vel_x: 0.5
3   min_vel_x: -0.5
4   max_vel_theta: 4.25
5   min_vel_theta: -4.25
6   min_in_place_vel_theta: 0.0
7
8   acc_lim_theta: 10.0
9   acc_lim_x: 10.0
10  acc_lim_y: 0.0
11  escape_vel: -0.5
12
13  holonomic_robot: false
14  pdist_scale: 1.5 # How close it follows global plan
15  # gdist_scale: 2.0 # How much it follows a local goal (0.8, max 5.0)
16  occdist_scale: 0.01 # How much it avoids obstacles
17  yaw_goal_tolerance: 0.2
18  xy_goal_tolerance: 0.2
```

```
19   dwa: false
20
21 GlobalPlanner:
22   use_dijkstra: true
```

planners_params.launch

En el caso del planificador basado en bandas elásticas temporales, su configuración se diferencia en los parámetros del *planner*.

```
1 TebLocalPlannerROS:
2
3   # Robot
4   acc_lim_x: 10.0
5   acc_lim_y: 0.0
6   acc_lim_theta: 10.0
7   max_vel_x: 0.5
8   max_vel_x_backwards: 0.5
9   max_vel_y: 0.0
10  max_vel_theta: 4.25
11  # allow_init_with_backwards_motion: true
12  footprint_model/type: "circular"
13  footprint_model/radius: 0.174
14
15  # HCPlanning
16  max_number_classes: 2 # Default: 5
17
18  # Optimization
19  no_inner_iterations: 2 # Default: 5
20  no_outer_iterations: 2 # Default: 5
21  weight_acc_lim_x: 0.0
22  weight_acc_lim_y: 0.0
23  weight_acc_lim_theta: 0.0
24
25  # Trajectory
26  dt_ref: 0.1
27  dt_hysteresis: 0.02
28
29  # Obstacles
30  min_obstacle_dist: 0.1 # Default: 0.5
```

```
31    inflation_dist: 0.5 # Default: 0.6
32    costmap_obstacles_behind_robot_dist: 0.5 # Default: 1.5
33    obstacle_poses_affected: 10 # Default: 30
```

local_planners_params.launch

Antes de proceder con la navegación autónoma real, se diseña un entorno virtual inspirado en el mapa construido en apartado 4.5. Esto sirve como mesa de pruebas para configurar los planificadores y decantarse por uno de ellos para su implementación. En las figuras 4.6.2 y 4.6.3 se comparan el mapa virtual con el real.

Figura 4.6.2: Mapa real (SLAM)

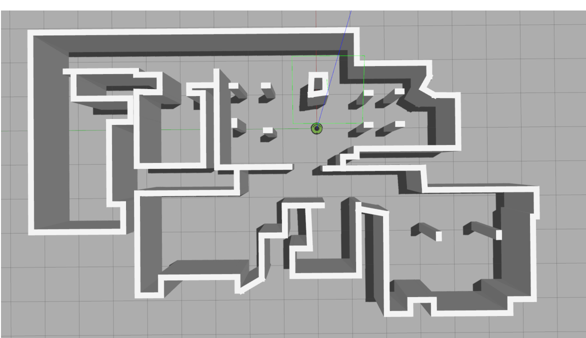


Figura 4.6.3: Entorno virtual

En la experiencia obtenida, modificar los algoritmos globales no tiene un impacto tan considerable como los planificadores locales, por lo que se han realizado pruebas con estos últimos. Debido a los huecos relativamente pequeños de las puertas, los algoritmos locales juegan un papel esencial en la velocidad con la que aseguran que se franquea estas puertas respetando las distancias respecto a las paredes. Se han probado tres opciones: *Trajectory Rollout*, *Dynamic Window Approach* y bandas elásticas temporales Este último gracias al paquete *teb_local_planner* (Rösmann, 2020). Este

nodo proporciona una interfaz altamente configurable, por lo que ofrece una gran flexibilidad para adecuarla al entorno y rendimiento deseado.

Las pruebas consisten en recorrer un circuito por el mapa con varios objetivos y cronometrando el tiempo invertido en conseguirlos (figura 4.6.4).

Figura 4.6.4: Visualización del la navegación en entorno virtual

El código se encuentra en *benchmark.py* en el anexo. Los resultados se exponen en la figura 4.6.5.

Por ello, se opta por el planificador local basado en bandas elásticas temporales. Además, ofrece una respuesta más rápida y eficaz en la evitación de obstáculos, algo de gran importancia en la navegación multi-robot.

4.7. Navegación autónoma real

En lo que respecta a la implementación real de la navegación autónoma, se ha procedido de manera similar al apartado anterior. Uno de los mayores problemas a los que se ha hecho frente es el de los retrasos en la comunicación. Ya que el procesamiento de datos se hace de manera remota, el envío y recepción de la información hace que los tiempos de margen para considerarse válidas las posiciones detectadas por el nodo de localización no sean válidas. Por ello, la solución propuesta ha sido aumentar la tolerancia de la publicación de esta posición. Incrementando el parámetro *tranform_tolerance* se posterga a futuro en dicha cantidad de tiempo la

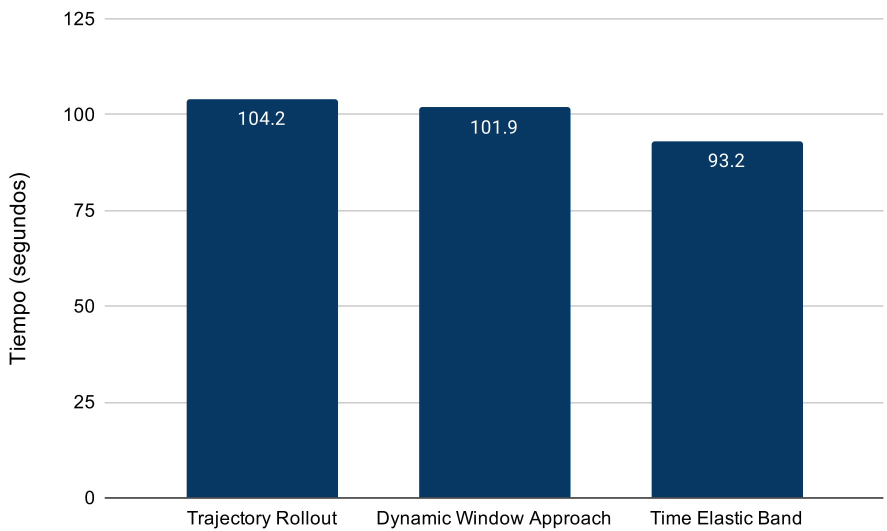


Figura 4.6.5: Comparativa de tiempos de navegación

transformada que informa de la posición del robot. Esto no reduce el retraso existente, pero hace factible la planificación y navegación.

4.8. Navegación multi-robot en simulación

Una vez puesta a punto la navegación autónoma, se procede con la navegación multi-robot. La forma de lanzar los nodos es similar a lo expuesto en apartados previos, por lo que no se incluye el código. Cabe señalar que es necesario un especial cuidado a la hora de especificar los espacios de trabajo (*namespaces*), ya que en ocasiones no basta con incluir los nodos bajo un espacio concreto y hay que proporcionar los *namespaces* como argumentos en las llamadas a archivos. En la simulación, debido al coste computacional de Gazebo y los cálculos de los algoritmos de navegación, se ha reducido la velocidad de simulación y se ha aumentado el tamaño de paso máximo del motor de físicas. Con ello, se consigue dejar más tiempo para el procesamiento a costa de un mayor tiempo de simulación.

Para asegurar que la navegación es segura en todo momento, se ha probado en simulación (*p5_multinav.launch*) que los robots son capaces de gestionar de forma eficaz un encuentro con otro u otros robots. Estos encuentros son considerados obstáculos dinámicos y el robot define una nueva trayectoria de manera local y sin saber la posición de los otros agentes. En las siguientes figuras se representan con colores distintos las trayectorias planificadas para cada robot. En una tonalidad más oscura se muestran las trayectorias globales mientras que se usa un color más claro

para los caminos planificados por el algoritmo local. Este último es el que lleva a cabo el sorteo de obstáculos recalculando una nueva ruta local. Además, se incluye el mapa de coste de uno de los robots para visualizar cómo percibe los obstáculos del entorno. El color azul celeste representa los espacios con un coste máximo y que no puede recorrer, siendo dicho coste menor en las franjas rojas y moradas. En primer lugar se simula la situación de un cruce frontal entre dos robots (figura 4.8.1).

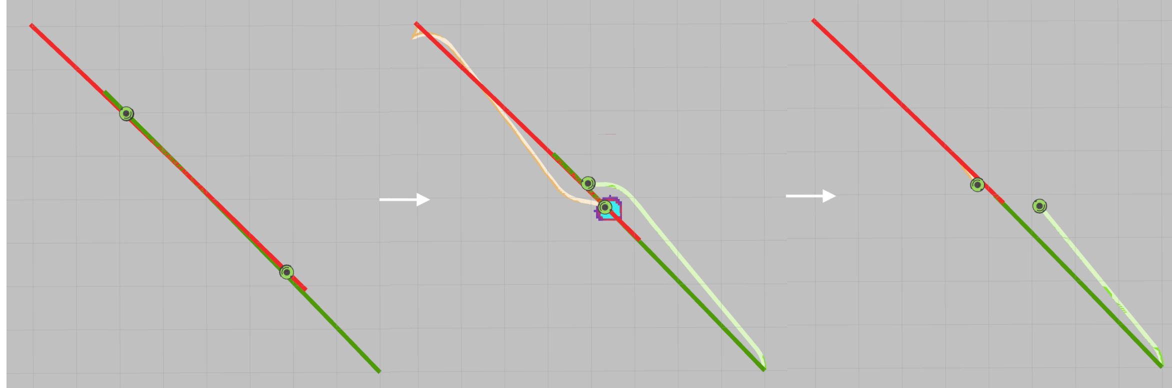


Figura 4.8.1: Cruce frontal

Posteriormente, se procede con un cruce a noventa grados entre dos robots (figura 4.8.2).

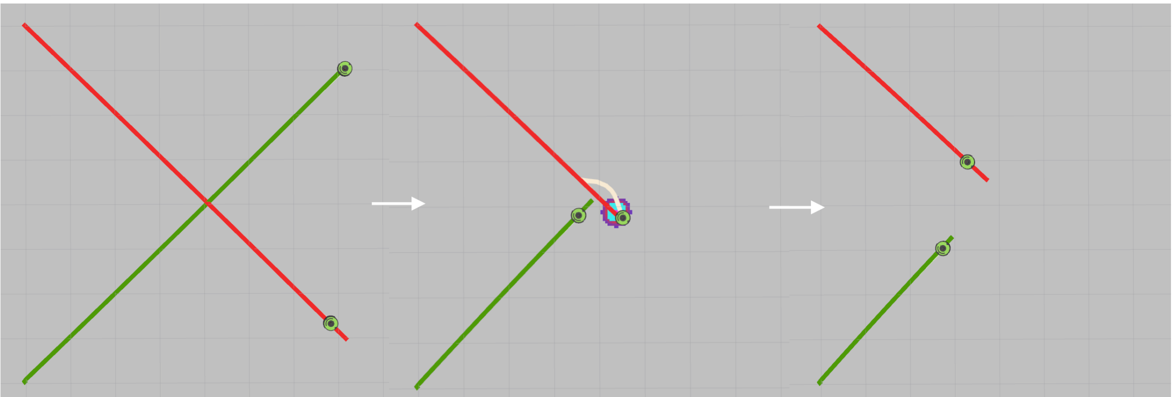


Figura 4.8.2: Cruce a noventa grados

Y finalmente, un cruce frontal de cuatro robots (figuras 4.8.3 y 4.8.4). Esta situación no es muy probable, pero el sistema debe ser robusto ante cualquier casuística y se debe probar su capacidad para solventar este tipo de problemas.

Como se aprecia, las rutas no son las óptimas, ya que no son gestionadas por un procesador central sino local, pero aseguran la robustez frente a fallos. Los comandos de posición se publican con el nodo propio (*goal_pub.py*).

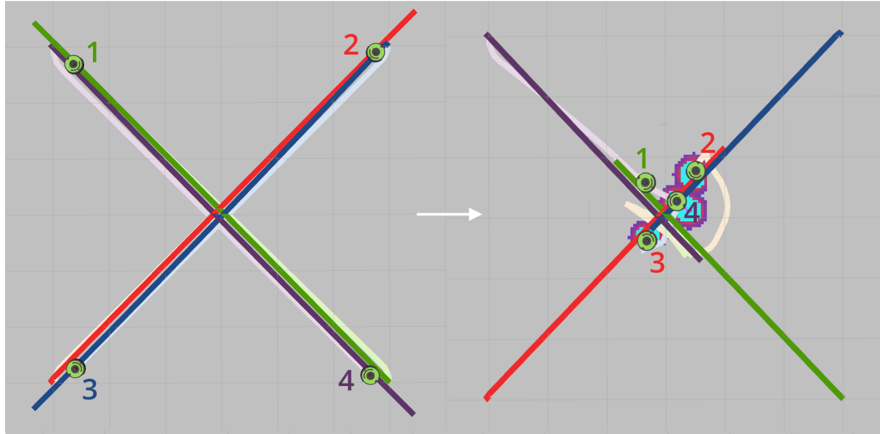


Figura 4.8.3: Cruce frontal de cuatro robots (inicio)

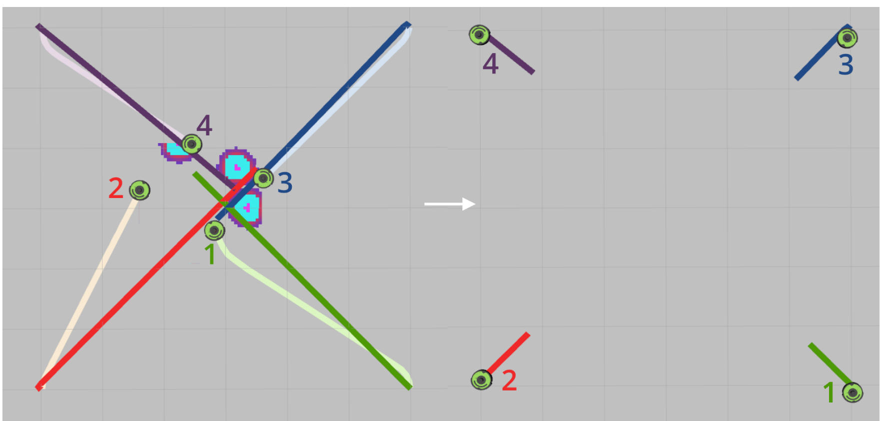


Figura 4.8.4: Cruce frontal de cuatro robots (final)

4.9. Navegación multi-robot real

Tras comprobar el correcto funcionamiento en simulación, se ha implementado en el robot real. En el mundo real no se puede reducir la velocidad de simulación, por lo que se debe limitar todo lo posible el coste computacional que supone planificar para dos robots en paralelo.

En primer lugar, se ha elaborado un mapa más pequeño, de una sola habitación y con una resolución menor (figura 4.9.1).

El siguiente paso es reducir la ventana del mapa de coste local hasta un metro alrededor del robot. En cuanto al planificador local, se aumenta el tamaño de paso del plan calculado (dt_ref) y se reduce el número de llamadas al solver (*no_inner_iterations* y *no_outer_iterations*). Siguiendo el mismo planteamiento, se reducen el horizonte futuro de planificación (*max_global_plan_lookahead_dist*) y el número de clases

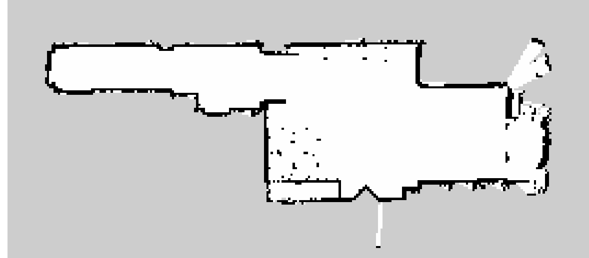


Figura 4.9.1: Mapa nuevo

homotópicas alternativas (*max_number_classes*). En topología algebraica, dos aplicaciones son homotópicas si una de ellas puede deformarse continuamente en la otra. Estas clases homotópicas determinan, por ejemplo, por donde se debe sortear un obstáculo (derecha o izquierda). Como referencia, en (Kuderer et al., 2014) y (Kala, 2016). se tratan la generación en linea de distintas trayectorias homotópicas para navegación autónoma.

A continuación se analiza el comportamiento mostrado por los robots (figuras 4.9.2 a 4.9.5). Primero, tras comprobar que los destinos impuestos mediante el nodo *goal_pub .py* son accesibles, se planea la trayectoria para cada uno de lo robots.

Figura 4.9.2: Cruce frontal RVIZ (1)

Figura 4.9.3: Cruce frontal real (1)

Tras reconocerse mutuamente como obstáculos en el mapa de coste, reducen su velocidad y se apartan de la trayectoria del otro. Uno de ellos incluso da marcha atrás al aproximarse excesivamente el otro robot. Un detalle a analizar aquí es el pequeño desfase entre los mapas de coste generados por los robots y sus posiciones reales. Esto se debe al retardo debido a la comunicación y de cómputo en paralelo de ambos. Estas trayectorias, a pesar de procesarse en el mismo equipo, son totalmente independientes y descentralizadas, y la única razón es la falta de potencia en los equipos remotos en cada robot. En una puesta en marcha real, fuera de los propósitos académicos de este proyecto, cada robot procesaría sus nodos de navegación.

La solución planteada pasa por aumentar el radio de inflación, así como la distancia mínima de separación entre obstáculos.

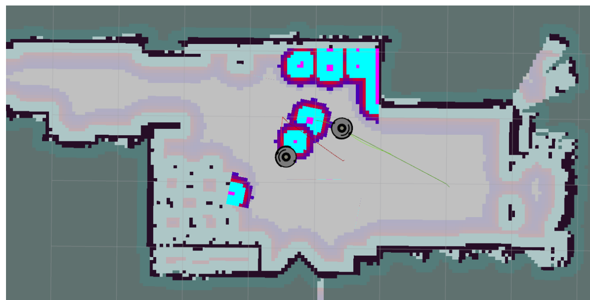


Figura 4.9.4: Cruce frontal RVIZ (2)

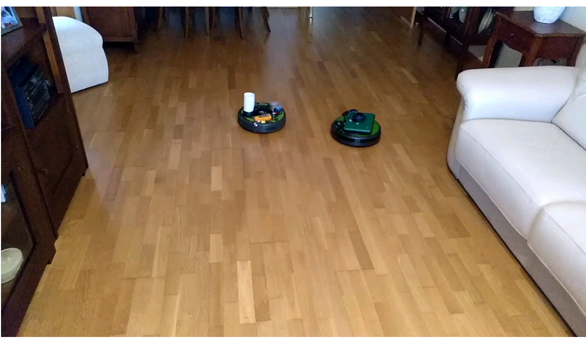


Figura 4.9.5: Cruce frontal real (2)

Como se observa, los robots son capaces de evitarse mutuamente y llegar a su destino.

4.10. Planificación multi-robot

La planificación multi-robot se ha enfocado desde un punto de vista dinámico, en el que el reparto de cada una de las tareas a realizar se hará en tiempo de ejecución y

teniendo en cuenta la viabilidad e idoneidad de cada robot para desempeñarla. Dichas tareas consisten en un "transporte" entre un punto inicial y final. Cada tarea está definida por sus puntos de recogida y entrega, no intercambiables con otras tareas, y se asignarán a los robots de forma dinámica.

La implementación en ROS se ha confeccionado nuevamente en Python mediante los nodos *auctioneer.py* y *bidder.py*, correspondientes al subastador y licitador. La parte del código del subastador que se describe en este apartado comienza con la definición de la clase *Auction* que proveerá todas las funciones para desarrollar el algoritmo. En este fragmento se definen las posiciones de las tareas, así como el número de robots, las variables contenedor para las apuestas y estados, y el tiempo de subasta. Las posiciones se han escogido de forma que ocurran cruces y encuentros de varios agentes para demostrar la habilidad del sistema para evitar choques y replanificar rutas.

```
17 class Auction():
18     def __init__(self, n_robots, robot_ns, auction_time):
19         self.x1 = [1.0, 0.0, -0.5, 0.25, 1.0, -0.5, 0.25, -0.75]
20         self.y1 = [1.0, -1.0, 0.0, 0.25, 1.0, -1.0, -0.25, -0.25]
21         self.x2 = [-0.25, 1.0, 0.0, 0.0, 0.5, -0.5, 0.0, 1.0]
22         self.y2 = [-0.25, 1.5, 1.5, -1.25, -1.25, 1.25, 1.0, 1.75]
23 
24         self.n_tasks = len(self.x1)
25         self.done_tasks = 0
26         self.n_robots = n_robots
27         self.bids = [-1]*n_robots
28         self.robot_ns = robot_ns
29         self.states = [-1]*n_robots
30         self.auction_time = auction_time
```

auctioneer.py

A continuación, se definen los tópicos sobre los que se publican las subastas y sus resultados. De forma similar, se suscribe a los tópicos donde se reciben las apuestas y las señales de disponibilidad de los robots.

```
31         for i in range(1, self.n_robots + 1):
32             rospy.Subscriber(str(robot_ns) + str(i) + '/bid', Bid,
                   self.bid_callback, i-1)
33 
```

```
34        for i in range(1, self.n_robots + 1):
35            rospy.Subscriber(str(robot_ns) + str(i) + '/bidder_ready',
36                             Int16, self.state_callback, i-1)
```

auctioneer.py

Se definen las funciones de retorno de llamada, para gestionar los mensajes entrantes.

```
37    def bid_callback(self, msg, i):
38        self.bids[i] = msg.bid
39
40    def state_callback(self, msg, i):
41        self.states[i] = msg.data
```

auctioneer.py

La parte fundamental del código es la siguiente. En primer lugar se publica la subasta y se espera el tiempo determinado por *self.auction_time*. Tras ello, se comprueba cual es la mejor apuesta (menor tiempo) y qué robots no han pujado, para identificar posibles inoperatividades o fallos de comunicación. Posteriormente, se publica el encargo con el identificador de robot que lo debe gestionar.

```
42    def start_auction(self):
43        for i in range(1, self.n_robots + 1):
44            if auction.done_tasks < auction.n_tasks:
45                self.bids = [-1]*n_robots
46                task = Task()
47                task.x1 = self.x1[self.done_tasks]
48                task.y1 = self.y1[self.done_tasks]
49                task.x2 = self.x2[self.done_tasks]
50                task.y2 = self.y2[self.done_tasks]
51                self.auction_pub.publish(task)
52                time.sleep(self.auction_time)
53                print(self.bids)
54
55                min_bid = 1e6
56                min_bid_id = -1
57                for j in range(1, self.n_robots + 1):
58                    bid = self.bids[j-1]
```

```
59                  if bid == -1: # If robot has not bid
60                      print(str(self.robot_ns) + str(j) + ' has not bid')
61                  else:
62                      if bid < min_bid:
63                          min_bid = bid
64                          min_bid_id = j
65
66              if min_bid_id != -1:
67                  result = AuctionResult()
68                  result.task = task
69                  result.robot_id = str(self.robot_ns) + str(min_bid_id)
70                  self.result_pub.publish(result)
71                  self.done_tasks += 1
```

auctioneer.py

Por último, se vuelve al inicio de programa conocido de otros nodos propios. En este caso, se pueden configurar por comando desde el terminal los parámetros de *namespaces* de los robots o modificar el número de éstos. Un parámetro significativo que se establece al inicio del código es el *max_n_robots_idle*, que hace referencia al número máximo de robots que se permite que estén inactivos simultáneamente. Si existiera un número mayor de agentes a la espera, se comenzaría una nueva subasta. Una vez se han completado todos los encargos, se muestra por terminal el tiempo invertido en completarlos, que servirá como resultado para un posterior análisis y comparativa.

```
72 if __name__ == '__main__':
73     rospy.init_node('auctioneer')
74     rate = rospy.Rate(5.0)
75
76     print('Auctioneer ready')
77     print('')
78
79     if len(sys.argv) > 1:
80         robot_ns = sys.argv[1]
81     if len(sys.argv) > 2:
82         n_robots = int(sys.argv[2])
83
84     auction = Auction(n_robots, robot_ns, auction_time, real)
85     finished = False
```

```
86
87     while (not rospy.is_shutdown()) and (not finished):
88         if auction.done_tasks < auction.n_tasks:
89             if auction.states.count(1) > max_n_robots_idle:
90                 if auction.done_tasks == 0:
91                     t_init = rospy.get_time()
92                 auction.start_auction()
93
94         else:
95             if auction.states == [1]*n_robots:
96                 finished = True
97                 t_end = rospy.get_time()
98                 print('')
99                 print('Total time: ' + str(t_end-t_init) + ' seconds')
100
101        rate.sleep()
```

auctioneer.py

En lo que respecta al código del licitador, se define la clase Robot() de forma análoga al subastador, indicando los nombres de los tópicos de suscripción y publicación. En este caso, dado que se diseña para ser lanzado desde un archivo *.launch*, el espacio de trabajo viene dado por defecto, por lo que no será necesario indicarlo y solo serán tópicos con referencia absoluta para los correspondientes al subastador.

```
15 class Robot():
16     def __init__(self, robot_ns):
17         self.robot_ns = robot_ns
18         self.pose = Pose()
19         self.velocity = 0.4
20         self.state = -1
21         self.ready = 0
22
23         self.bid_pub = rospy.Publisher('bid', Bid, queue_size=1)
24         self.move_pub = rospy.Publisher('move_base_simple/goal',
               PoseStamped, queue_size=1)
25         self.ready_pub = rospy.Publisher('bidder_ready', Int16,
               queue_size=1)
26
```

```
27        rospy.Subscriber('amcl_pose', PoseWithCovarianceStamped,
             self.pose_callback)
28        rospy.Subscriber('/auction', Task, self.task_callback)
29        rospy.Subscriber('/auction_result', AuctionResult,
             self.result_callback)
30        rospy.Subscriber('move_base/status', GoalStatusArray,
             self.state_callback)
```

bidder.py

En cuanto a las funciones de retorno de llamada, se ha hecho una simplificación para ahorrar tiempo en el procesamiento. A la hora de calcular la puja de cada robot, la distancia que debería recorrer se ha limitado a la distancia euclídea que separa a los puntos. En un caso real, esta distancia se calcularía enviando el destino al planificador e integrando los segmentos que componen la ruta propuesta. Dado que en la simulación, y el caso real que se tratará en el siguiente apartado, el espacio es convexo, las aproximaciones son menores.

```
31    def pose_callback(self, msg):
32        self.pose = msg.pose.pose
33
34    def task_callback(self, msg):
35        if self.ready:
36            distance = (math.sqrt((msg.x2-msg.x1)**2 +
                 (msg.y2-msg.y1)**2) +
                 math.sqrt((msg.x1-self.pose.position.x)**2 +
                 (msg.y1-self.pose.position.y)**2))
37
38            self.bid_pub.publish(distance/self.velocity)
39
40    def state_callback(self, msg):
41        if len(msg.status_list) == 0:
42            self.state = -1
43        else:
44            self.state = msg.status_list[-1].status
```

bidder.py

La tarea a realizar consiste en cubrir dos trayectorias, por lo que primero se debe completar el traslado al punto de recogida y posteriormente el transporte al punto de entrega, antes de indicar su disponibilidad de nuevo.

```
45    def result_callback(self, msg):
46        if msg.robot_id == self.robot_ns:
47            self.ready = 0
48            self.ready_pub.publish(0) # Not ready
49            goal = self.calculate_goal(msg.task.x1, msg.task.y1)
50            self.move_pub.publish(goal)
51            time.sleep(wait_time_check_state)
52            while self.state != 3:
53                time.sleep(0.2)
54
55            goal = self.calculate_goal(msg.task.x2, msg.task.y2)
56            self.move_pub.publish(goal)
57            time.sleep(wait_time_check_state)
58            while self.state != 3:
59                time.sleep(0.2)
60
61            self.ready = 1
62            self.ready_pub.publish(1)
```

bidder.py

De forma ilustrativa, se han supuesto cuatro puntos de recogida en la parte superior del mapa (línea continua) y cuatro de entrega en la parte inferior (línea punteada). Nuevamente, los destinos están cruzados con intención de probar la capacidad resolutiva del robot ante encuentros con otros robots. Como muestra de este reparto de tareas, se adjunta la figura 4.10.1. A la izquierda, los robots se encuentran realizando dicho transporte desde el punto de recogida hasta el de entrega. Cuando finalizan, a la derecha, se reparten nuevamente la tarea de transporte que tengan más cercana, identificándose por colores según el robot que la ejecuta.

Además, también se ha probado su rendimiento ante puntos de recogida y entrega aleatorios en el mapa, con una mayor posibilidad de encuentros múltiples (figura 4.10.2).

Por último, para conseguir una mayor semejanza con una posible implemetación real, se ha recreado en Gazebo un entorno similar al de un invernadero para validar la

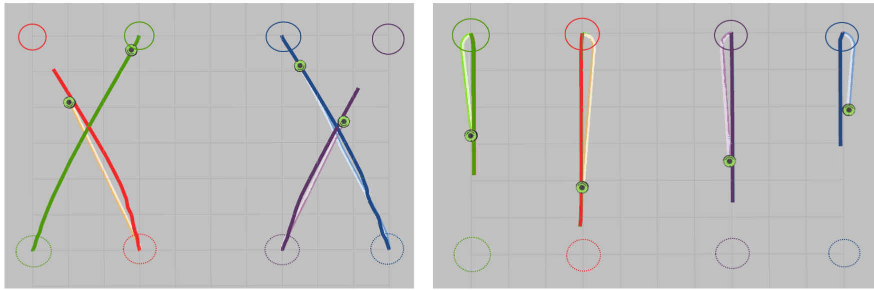


Figura 4.10.1: Planificación multi-robot en simulación (1)

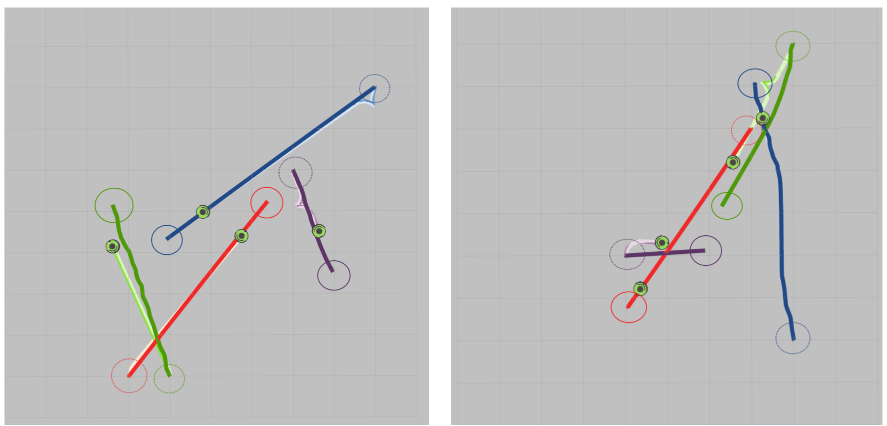


Figura 4.10.2: Planificación multi-robot en simulación (2)

planificación y navegación multi-robot en él (figura 4.10.3). El resultado de esta simulación se puede ver en (ARM UAL, 2020c).

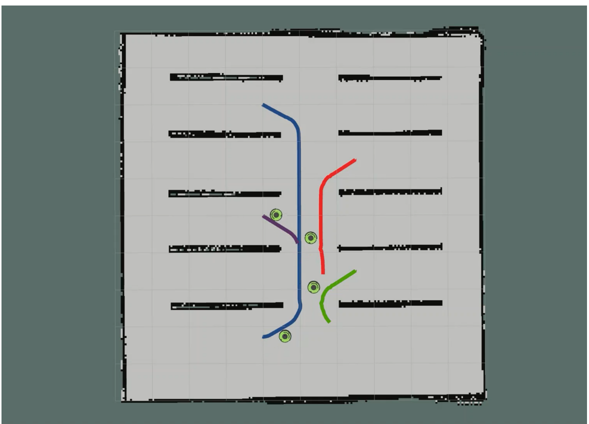


Figura 4.10.3: Planificación multi-robot en invernadero

Se han realizado pruebas variando el número de robots en el sistema, cuyos resultados se muestran a continuación. En la figura 4.10.4 se refleja el incremento de la velocidad de trabajo que tiene el sistema con el aumento de robots. De manera

similar, en la figura 4.10.5 se observa la reducción en la distancia total recorrida por el sistema.

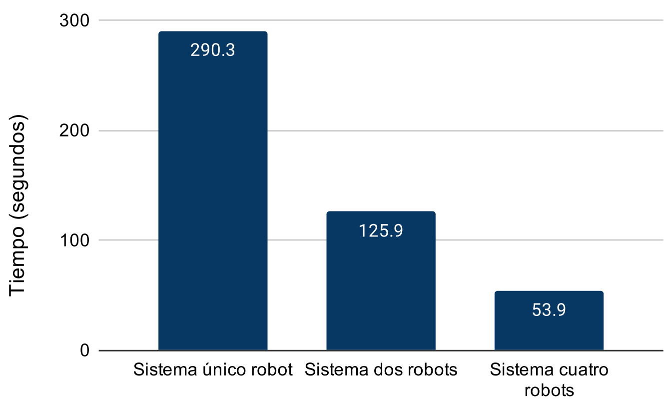


Figura 4.10.4: Comparativa de tiempos en simulación

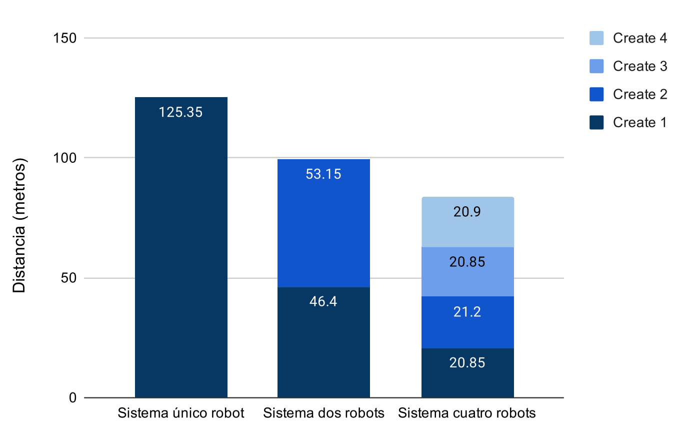


Figura 4.10.5: Comparativa de distancia recorrida en simulación

Estos datos señalan que el rendimiento de la flota de robots es mayor que su proporción correspondiente, es decir, un robot tarda más de cuatro veces en realizar el trabajo de cuatro robots. Esto se debe a que los objetivos se han planteado como tareas de transporte, con dos destinos en cada una (recogida y entrega). Esto hace que cuanto mayor sea el número de robots repartidos por el mapa, la distancia del robot más próximo al siguiente objetivo sea menor y por ende, menor tiempo empleado. También cabe destacar que esta proporción en la reducción de tiempo se vería mermada con un número muy elevado de agentes en el sistema, debido a las interferencias entre sus trayectorias.

4.11. Planificación multi-robot real

Una vez explicados todos los nodos (localización, navegación, ...) que intervienen en el sistema multi-robot, se expone en la figura 4.11.1 de forma muy resumida la relación y comunicación existente entre todos ellos. En color aparecen los nodos, representados con elipses, y tópicos, mediante cuadros, ejecutados en los robots móviles. Estos corresponden a los que llevan a cabo la interfaz con los dispositivos a bordo y la publicación de la información de los sensores. El resto de nodos son ejecutados por un ordenador central debido a su mayor capacidad de cómputo.

En una futura versión comercial del sistema, todos los nodos se ejecutarían en las unidades móviles, como se indicó en el apartado 4.10.

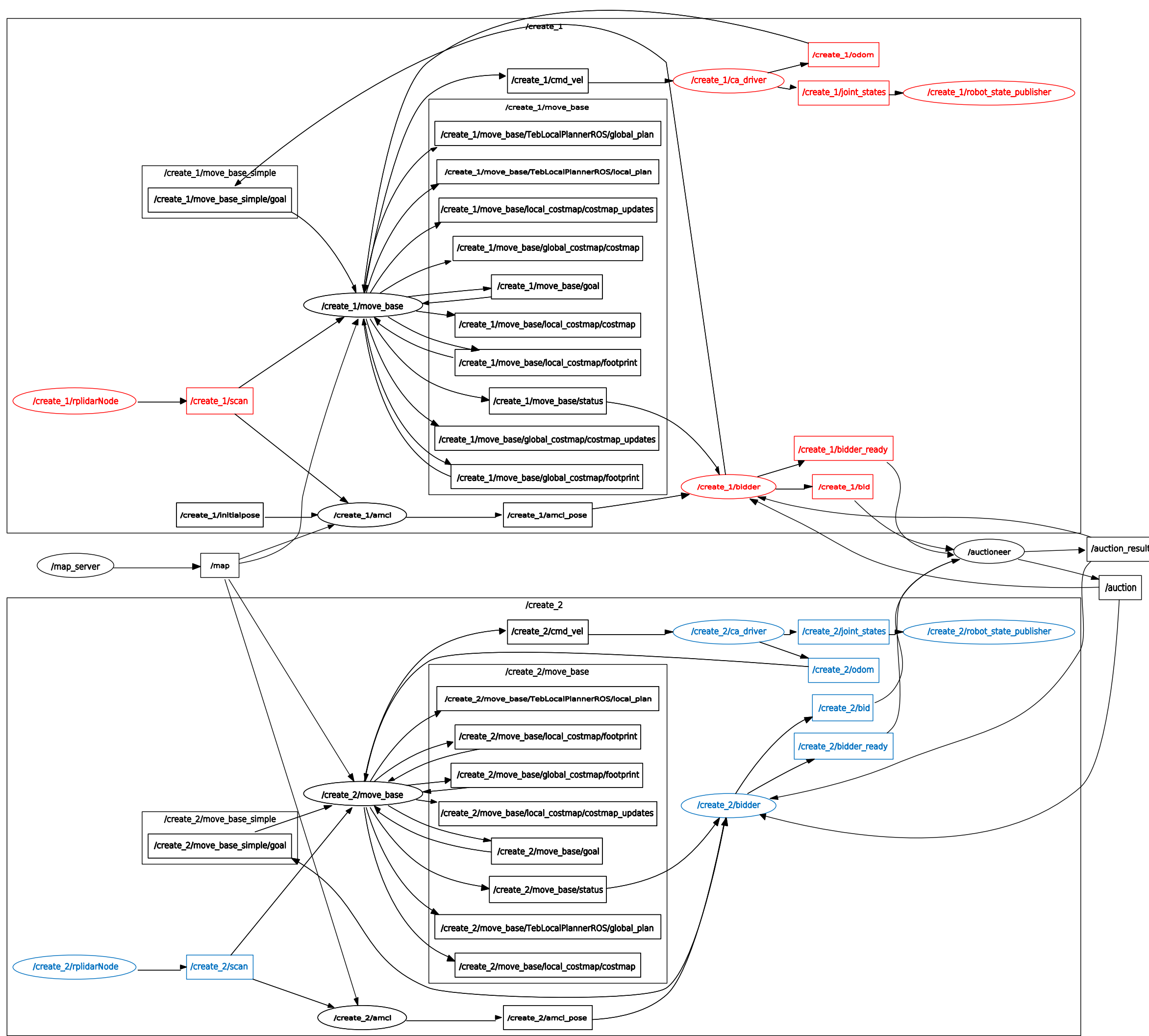


Figura 4.11.1: Estructura de nodos del sistema multi-robot

Finalmente, la transferencia al mundo real se ha realizado mediante dos pruebas en las que se compara el tiempo invertido en ejecutar ocho tareas de transporte en varios puntos del mapa. En la primera de ellas, sólo un robot se encarga de llevarlas a cabo, mientras que en la segunda se dispone de los dos Create para ello. Los puntos de recogida y entrega se muestran en las figuras 4.11.2 y 4.11.3.

Estos puntos se han ubicado en caminos cruzados para poner a prueba la navegación simultánea y para replicar de la forma más similar posible las pruebas realizadas en simulación. En este caso, debido a las restricciones de espacio, los robots invierten algo más de tiempo en esquivarse. A pesar de ello, consiguen llevar a cabo las tareas sin contacto entre ellos.

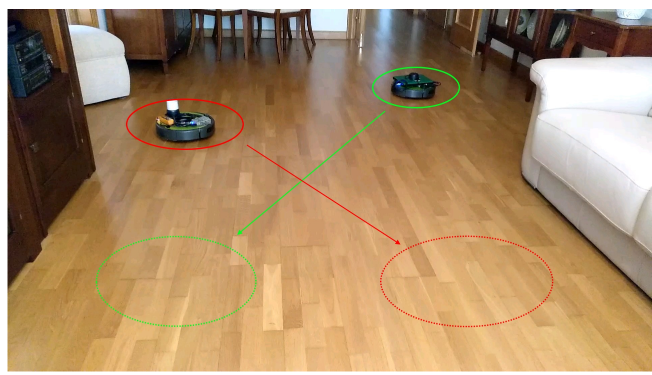


Figura 4.11.2: Puntos de recogida

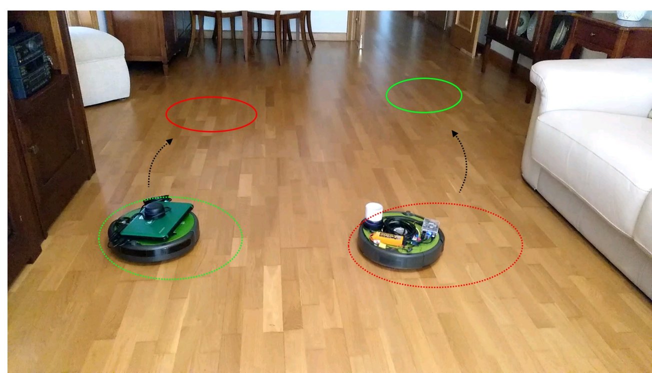


Figura 4.11.3: Puntos de entrega

El desempeño del sistema se puede visualizar en el vídeo disponible en el canal de Youtube del grupo de investigación ARM (ARM UAL, 2020a). Los resultados cuantitativos de estas pruebas se reflejan en las figuras 4.11.4 y 4.11.5. Se comprueba que el tiempo y la distancia se reducen a la mitad cuando se duplica el número de robots en el sistema. De esta forma, se valida la implementación del sistema real.

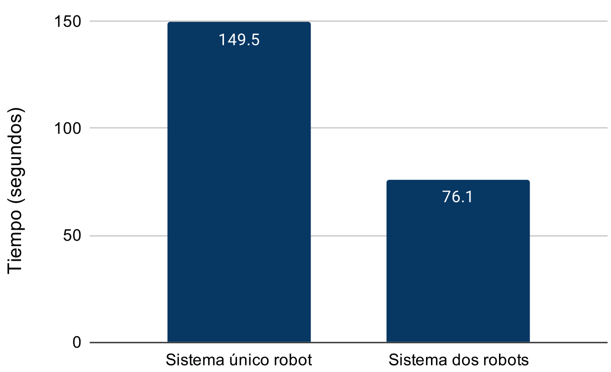


Figura 4.11.4: Comparativa de tiempos

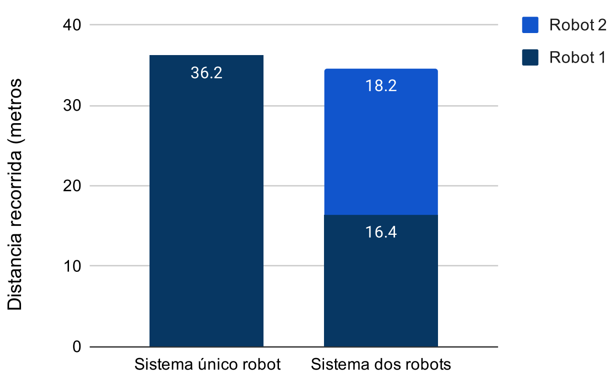


Figura 4.11.5: Comparativa de distancia recorrida

De manera adicional, y con el objetivo de mostrar las capacidades del sistema construido, se ha grabado otro vídeo para ilustrar su rendimiento con obstáculos dinámicos, disponible en (ARM UAL, 2020b).

Capítulo 5

Conclusiones y trabajos futuros

En este trabajo se ha demostrado cómo un sistema multi-robot es capaz de ejecutar una serie de tareas de forma estructurada y ordenada, permitiendo multiplicar la capacidad de trabajo en comparación con la utilización de un único robot. Esta tarea, aparentemente compleja en el inicio, se ha dividido en una serie de subobjetivos abordables, que resueltos de forma separada permiten conseguir un sistema robusto y completamente autónomo.

A lo largo de todo el proyecto se han superado una serie de retos de diversa índole. El problema del mapeado ha resultado ser uno de los más asequibles, por su rápida implementación mediante ROS, tanto en simulación como en el robot real. Los mapas construidos del entorno real son lo suficientemente exactos para permitir una localización precisa y rápida del robot. La resolución de los mismos se ha reducido en la navegación multi-robot para aliviar tiempo de cómputo y mantener una mayor frecuencia de actualización.

Sin embargo, la configuración de la navegación autónoma ha sido un proceso largo y tedioso. En espacios abiertos, la mayoría de planificadores devuelven una solución rápida y efectiva, pero esto no es el caso cuando el robot discurre por pasillos estrechos o cruza los marcos de las puertas. Esta ha sido una de las tareas en las que más tiempo se ha invertido, y cuyos resultados no han sido siempre satisfactorios. Los parámetros del planificador afectan de manera notable a su rendimiento, por lo que ha sido necesario un ajuste cauteloso. A su vez se debía lidiar con los problema de la limitación de capacidad de procesamiento y retardos en las comunicaciones, como se ha expuesto en el apartado 4.9. Por ello, la sintonización adecuada ha sido fundamental para obtener un rendimiento adecuado pero con la suficiente tasa de refresco para detectar a tiempo los obstáculos dinámicos en la navegación multi-robot.

Un planteamiento inicial para abordar la navegación multi-robot fue el uso de redes neuronales, como se plantea en (Long et al., 2018; Tan et al., 2019). Se han realizado pruebas con la biblioteca Tensorflow (Google LLC, s.f.) en ROS y Gazebo, pero los resultados no fueron los esperados. No obstante, esta línea de investigación es una de las que tienen un mayor auge en la actualidad, por lo que esta es una de las principales vías abiertas para trabajos futuros.

De igual forma, la cooperación entre los diferentes robots está ampliamente estudiada con técnicas basadas en aprendizaje por refuerzo (*reinforcement learning*). Mediante este enfoque se permite optimizar un grupo de agentes numeroso para tareas de una mayor complejidad de manera descentralizada, como se presenta en (Gupta, Egorov y Kochenderfer, 2017).

Por otra parte, la distribución de tareas entre los diferentes agentes del sistema se ha realizado con una implementación propia basada en el enfoque de subasta y puja. Esta programación es eficiente en los casos probados, con tareas relativamente sencillas y el mismo tipo de robot. Sin embargo, se podría extender a procesos con robots heterogéneos en los que las capacidades de cada uno exijan una mayor optimización de dicho reparto, como las propuestas de los trabajos (Shi et al., 2010; Tang y Parker, 2005).

Anexos

A. Presupuesto del proyecto

A continuación se detalla el presupuesto global del material empleado en el proyecto en el tabla A.1.

Material	Unidades	Precio unitario (€)	Precio total (€)
iRobot Create 2	2	271.04	542.08
Slamtec RPLIDAR A3M1	2	567.49	1134.98
Raspberry Pi 3B+	2	32.19	64.38
Fuente de alimentación 5V 3A	1	12.99	12.99
Memoria usb Sandisk Ultra Flair USB 3.0	2	12.90	25.80
Convertidor buck DC-DC XL4015 5A	2	1.75	3.50
Batería LiPo U-TECH PRO 3S 11.1 V 2200mAH	2	17.77	35.54
Conector XT60 batería 5 pares	1	2.11	2.11
Conector macho micro USB 10 unidades	1	1.45	1.45
Abrazaderas cable clip 10 unidades	1	1.11	1.11
Cables 22 AWG	3	0.69	2.07
Interruptores 5 unidades	1	1.38	1.38
Mano de obra (horas)	420	25.00	10500.00
Portátil Acer Aspire F15	1	669.90	669.90
			12997.29

Tabla A.1: Presupuesto del proyecto

B. Presupuesto comercial

El presente trabajo tiene como principal objetivo la implementación de un sistema real con una posible aplicación comercial futura. Como se refleja en el apartado anterior, el coste material completo de un robot sería de 913,70 euros. Suponiendo una mano de obra para la puesta en funcionamiento de 10 horas por cada robot, una media de 5

robots por cada sistema instalado y un beneficio industrial del 8 %, se conseguiría una amortización de la inversión del proyecto tras 35 sistemas vendidos. Estas cifras son meramente orientativas, ya que, enmarcado en un contexto académico, la mayor parte de esa inversión ha sido en tiempo dedicado a adquirir los conocimientos necesarios para llevarlo a cabo.

C. Código fuente

A lo largo del proyecto se han creado un gran número de ficheros de código fuente. Sin embargo, se trata de archivos muy similares entre sí, sobre todo en el caso de parámetros de configuración. Por esta razón en esta sección de código sólo se incluirá una muestra de este tipo de archivos, para evitar redundancia.

C.1. Paquete tfg_simulations

Carpeta config

p2_base

```
1 obstacle_range: 5.0
2 raytrace_range: 5.0
3 robot_radius: 0.174
4 observation_sources: laser_scan_sensor
5 transform_tolerance: 0.1
6 laser_scan_sensor: {sensor_frame: lidar, data_type: LaserScan, topic:
     scan, marking: true, clearing: true}
```

costmap_common_params.yaml

```
1 local_costmap:
2   global_frame: odom
3   robot_base_frame: base_footprint
4   update_frequency: 5.0
5   publish_frequency: 1.0
6   static_map: false
7   rolling_window: true
8   width: 3.0
```

```
9   height: 3.0
10  resolution: 0.05
11  inflation_radius: 0.4
```

local_costmap_params.yaml

```
1 TrajectoryPlannerROS:
2   max_vel_x: 0.5
3   min_vel_x: -0.5
4   max_vel_theta: 4.25
5   min_vel_theta: -4.25
6   min_in_place_vel_theta: 0.0
7   acc_lim_theta: 10.0
8   acc_lim_x: 10.0
9   acc_lim_y: 0.0
10  escape_vel: -0.5
11  holonomic_robot: false
12  pdist_scale: 1.5 # How close it follows global plan (default: 0.6,
       max: 5.0)
13  # gdist_scale: 2.0 # How much it follows a local goal (0.8, max 5.0)
14  occdist_scale: 0.01 # How much it avoids obstacles (0.01)
15  yaw_goal_tolerance: 0.2
16  xy_goal_tolerance: 0.2
17  dwa: false
18
19 GlobalPlanner:
20  use_dijkstra: true
```

planners_params.yaml

p2_teb

```
1 TebLocalPlannerROS:
2   # Robot
3   acc_lim_x: 10.0
4   acc_lim_y: 0.0
5   acc_lim_theta: 10.0
6   max_vel_x: 0.5
7   max_vel_x_backwards: 0.5
```

```
  max_vel_y: 0.0
  max_vel_theta: 4.25
  # allow_init_with_backwards_motion: true
  footprint_model/type: "circular"
  footprint_model/radius: 0.174

  # HCPlanning
  max_number_classes: 2 # Default: 5

  # Optimization
  no_inner_iterations: 2 # Default: 5
  no_outer_iterations: 2 # Default: 5
  weight_acc_lim_x: 0.0
  weight_acc_lim_y: 0.0
  weight_acc_lim_theta: 0.0

  # Trajectory
  dt_ref: 0.1
  dt_hysteresis: 0.02

  # Obstacles
  min_obstacle_dist: 0.1 # Default: 0.5
  inflation_dist: 0.5 # Default: 0.6
  costmap_obstacles_behind_robot_dist: 0.5 # Default: 1.5
  obstacle_poses_affected: 10 # Default: 30
```

local_planner_params.yaml

p4_eband

```
EBandPlannerROS:
  max_vel_lin: 1.0
  max_vel_th: 1.5
  min_vel_lin: 0.0
  min_vel_th: 0.0
  min_in_place_vel_th: 0.0
  # max_acceleration: 10.0
  max_translational_acceleration: 15.0
  max_rotational_acceleration: 5.0
```

```
10  # Key parameters
11  eband_internal_force_gain: 1.0
12  eband_external_force_gain: 10.0 # Default 2.0
13  differential_drive: true
14  Ctrl_Rate: 10.0
```

base_local_planner_params.yaml

Carpeta launch

```
1 <launch>
2
3   <!-- Parameters configuration -->
4   <arg name="paused" default="false"/>
5   <arg name="use_sim_time" default="true"/>
6   <arg name="gui" default="true"/>
7   <arg name="headless" default="false"/>
8   <arg name="debug" default="false"/>
9
10  <!--Gazebo parameters -->
11  <include file="$(find gazebo_ros)/launch/empty_world.launch">
12    <arg name="world_name" value="$(find
          tfg_simulations)/worlds/p1_slam.world"/>
13    <arg name="debug" value="$(arg debug)" />
14    <arg name="gui" value="$(arg gui)" />
15    <arg name="paused" value="$(arg paused)"/>
16    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
17    <arg name="headless" value="$(arg headless)"/>
18  </include>
19
20  <!-- Set robot model parameter -->
21  <param name="robot_description"
22    command="$(find xacro)/xacro '$(find
          tfg_simulations)/urdf/create2.xacro'" />
23
24  <!-- Spawn robot model in Gazebo -->
25  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
      respawn="false" output="screen"
26    args="-urdf -model create -param robot_description"/>
```

```
27
28   <!-- Publishes joints state at certain rate -->
29   <node name="joint_state_publisher" pkg="joint_state_publisher"
         type="joint_state_publisher">
30     <param name="use_gui" value="false"/>
31     <param name="rate" value="20"/>
32   </node>
33
34   <!-- Publishes all the tf for the robot -->
35   <node name="robot_state_publisher" pkg="robot_state_publisher"
         type="robot_state_publisher">
36     <param name="publish_frequency" value="60.0"/>
37     <param name="use_tf_static" value="true"/>
38   </node>
39
40     <!-- SLAM mapping of the environment -->
41   <node name="mapping" pkg="gmapping" type="slam_gmapping"
         args="scan:=scan">
42     <param name="base_frame" value="base_link"/>
43     <param name="map_update_interval" value="0.5"/>
44   </node>
45
46   <!-- Custom nodes for PS3 controller commands -->
47   <node name="joystick_publisher" pkg="tfg_simulations"
         type="ps3_node.py" respawn="true" />
48   <node name="joystick_to_cmd" pkg="tfg_simulations"
         type="create_cmd.py"/>
49
50   <!-- Show everything in Rviz -->
51   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
         tfg_simulations)/config/p1_slam.rviz"
52     respawn="true"/>
53
54 </launch>
```

p1_slam.launch

```
1 <launch>
2
```

```
3   <!-- these are the arguments you can pass this launch file, for
       example paused:=true -->
4   <arg name="paused" default="false"/>
5   <arg name="use_sim_time" default="true"/>
6   <arg name="gui" default="true"/>
7   <arg name="headless" default="false"/>
8   <arg name="debug" default="false"/>
9
10  <!-- Select planner (ONLY ONE MUST BE TRUE) -->
11  <arg name="base_planner" value="false"/>
12  <arg name="teb_planner" value="true"/>
13
14  <!--Gazebo parameters -->
15  <include file="$(find gazebo_ros)/launch/empty_world.launch">
16    <arg name="world_name" value="$(find
        tfg_simulations)/worlds/p2_autonav.world"/>
17    <!-- <arg name="world_name" value="$(find
        tfg_simulations)/worlds/house.world"/> -->
18    <arg name="debug" value="$(arg debug)" />
19    <arg name="gui" value="$(arg gui)" />
20    <arg name="paused" value="$(arg paused)"/>
21    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
22    <arg name="headless" value="$(arg headless)"/>
23  </include>
24
25  <!-- Robot URDF / Gazebo description -->
26  <param name="robot_description"
27    command="$(find xacro)/xacro '$(find
        tfg_simulations)/urdf/create2.xacro'" />
28
29  <!-- Spawn robot model in Gazebo -->
30  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
      respawn="false" output="screen"
31    args="-urdf -model create -param robot_description"/>
32
33  <!-- Publishes joints state at certain rate -->
34  <node name="joint_state_publisher" pkg="joint_state_publisher"
      type="joint_state_publisher">
35    <param name="use_gui" value="false"/>
```

```
36      <param name="rate" value="20"/>
37    </node>
38
39    <!-- Publishes all the tf for the robot -->
40    <node name="robot_state_publisher" pkg="robot_state_publisher"
        type="robot_state_publisher"/>
41
42    <!-- Spawns a map of the environment -->
43    <node pkg="map_server" type="map_server" args="$(find
        tfg_simulations)/map/map.yaml"
44      respawn="true" name="map_server" />
45
46    <!-- Import move_base file -->
47    <include file="$(find tfg_simulations)/launch/p2_move_base.launch"
        if="$(arg base_planner)">
48      <arg name="base_planner" value="true" />
49    </include>
50    <include file="$(find tfg_simulations)/launch/p2_move_base.launch"
        if="$(arg teb_planner)">
51      <arg name="teb_planner" value="true" />
52    </include>
53
54    <!-- Show everything in Rviz -->
55    <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
        tfg_simulations)/config/autonomous_base.rviz"
56      respawn="true" if="$(arg base_planner)"/>
57    <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
        tfg_simulations)/config/autonomous_teb.rviz"
58      respawn="true" if="$(arg teb_planner)"/>
59
60  </launch>
```

p2_autonav.launch

```
1  <launch>
2
3    <arg name="base_planner" default="false"/>
4    <arg name="teb_planner" default="false"/>
5
```

```
6 <!-- Publish scans from best pose at a max of 10 Hz -->
7   <node pkg="amcl" type="amcl" name="amcl" output="screen"
        args="scan:=scan map:=map">
8     <param name="use_map_topic" value="false"/>
9     <param name="base_frame_id" value="base_footprint"/>
10    <param name="odom_model_type" value="diff"/>
11    <param name="odom_alpha5" value="0.1"/>
12    <param name="transform_tolerance" value="0.1" />
13    <param name="gui_publish_rate" value="10.0"/>
14    <param name="laser_max_beams" value="30"/>
15    <param name="min_particles" value="500"/>
16    <param name="max_particles" value="5000"/>
17    <param name="kld_err" value="0.05"/>
18    <param name="kld_z" value="0.99"/>
19    <param name="odom_alpha1" value="0.2"/>
20    <param name="odom_alpha2" value="0.2"/>
21    <!-- translation std dev, m -->
22    <param name="odom_alpha3" value="0.8"/>
23    <param name="odom_alpha4" value="0.2"/>
24    <param name="laser_z_hit" value="0.5"/>
25    <param name="laser_z_short" value="0.05"/>
26    <param name="laser_z_max" value="0.05"/>
27    <param name="laser_z_rand" value="0.5"/>
28    <param name="laser_sigma_hit" value="0.2"/>
29    <param name="laser_lambda_short" value="0.1"/>
30    <param name="laser_lambda_short" value="0.1"/>
31    <param name="laser_model_type" value="likelihood_field"/>
32    <!-- <param name="laser_model_type" value="beam"/> -->
33    <param name="laser_likelihood_max_dist" value="2.0"/>
34    <param name="update_min_d" value="0.2"/>
35    <param name="update_min_a" value="0.5"/>
36    <param name="odom_frame_id" value="odom"/>
37    <param name="resample_interval" value="1"/>
38    <param name="recovery_alpha_slow" value="0.0"/>
39    <param name="recovery_alpha_fast" value="0.0"/>
40  </node>
41
42  <!-- Base local planner -->
```

```
43  <node pkg="move_base" type="move_base" respawn="false"
        name="move_base_base" output="screen" if="$(arg base_planner)">
44    <rosparam file="$(find
          tfg_simulations)/config/p2_base/costmap_common_params.yaml"
          command="load" ns="global_costmap" />
45    <rosparam file="$(find
          tfg_simulations)/config/p2_base/costmap_common_params.yaml"
          command="load" ns="local_costmap" />
46    <rosparam file="$(find
          tfg_simulations)/config/p2_base/local_costmap_params.yaml"
          command="load" />
47    <rosparam file="$(find
          tfg_simulations)/config/p2_base/global_costmap_params.yaml"
          command="load" />
48    <rosparam file="$(find
          tfg_simulations)/config/p2_base/planners_params.yaml"
          command="load" />
49
50    <param name="controller_frequency" value="10.0"/>
51    <param name="base_local_planner"
          value="base_local_planner/TrajectoryPlannerROS" />
52    <param name="base_global_planner"
          value="global_planner/GlobalPlanner" />
53  </node>
54
55  <!-- TEB local planner -->
56  <node pkg="move_base" type="move_base" respawn="false"
        name="move_base_teb" output="screen" if="$(arg teb_planner)">
57    <rosparam file="$(find
          tfg_simulations)/config/p2_teb/costmap_common_params.yaml"
          command="load" ns="global_costmap" />
58    <rosparam file="$(find
          tfg_simulations)/config/p2_teb/costmap_common_params.yaml"
          command="load" ns="local_costmap" />
59    <rosparam file="$(find
          tfg_simulations)/config/p2_teb/local_costmap_params.yaml"
          command="load" />
```

```
60      <rosparam file="$(find
           tfg_simulations)/config/p2_teb/global_costmap_params.yaml"
           command="load" />
61      <rosparam file="$(find
           tfg_simulations)/config/p2_teb/local_planner_params.yaml"
           command="load" />
62
63      <param name="controller_frequency" value="10.0"/>
64      <param name="base_local_planner"
           value="teb_local_planner/TebLocalPlannerROS" />
65      <param name="base_global_planner"
           value="global_planner/GlobalPlanner" />
66    </node>
67
68 </launch>
```

p2_move_base.launch

```
1 <launch>
2
3   <!-- these are the arguments you can pass this launch file, for
       example paused:=true -->
4   <arg name="paused" default="false"/>
5   <arg name="use_sim_time" default="true"/>
6   <arg name="gui" default="true"/>
7   <arg name="headless" default="false"/>
8   <arg name="debug" default="false"/>
9
10  <!-- Select planner (ONLY ONE MUST BE TRUE) -->
11  <arg name="base_planner" value="true"/>
12  <arg name="teb_planner" value="false"/>
13
14
15  <!--Gazebo parameters -->
16  <include file="$(find gazebo_ros)/launch/empty_world.launch">
17    <arg name="world_name" value="$(find
         tfg_simulations)/worlds/p3_autonav.world"/>
18    <arg name="debug" value="$(arg debug)" />
19    <arg name="gui" value="$(arg gui)" />
```

```
20    <arg name="paused" value="$(arg paused)"/>
21    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
22    <arg name="headless" value="$(arg headless)"/>
23  </include>
24
25  <!-- Robot URDF / Gazebo description -->
26  <param name="robot_description"
27    command="$(find xacro)/xacro '$(find
        tfg_simulations)/urdf/create2.xacro'" />
28
29  <!-- Spawn robot model in Gazebo -->
30  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
      respawn="false" output="screen"
31    args="-urdf -model create -param robot_description"/>
32
33  <!-- Publishes joints state at certain rate -->
34  <node name="joint_state_publisher" pkg="joint_state_publisher"
      type="joint_state_publisher">
35    <param name="use_gui" value="false"/>
36    <param name="rate" value="20"/>
37  </node>
38
39  <!-- Publishes all the tf for the robot -->
40  <node name="robot_state_publisher" pkg="robot_state_publisher"
      type="robot_state_publisher"/>
41
42  <!-- Spawns a map of the environment -->
43  <node pkg="map_server" type="map_server" args="$(find
      tfg_simulations)/map/house_sim.yaml"
44    respawn="true" name="map_server" />
45
46  <!-- Import move_base file -->
47  <include file="$(find tfg_simulations)/launch/p3_move_base.launch"
      if="$(arg base_planner)">
48    <arg name="base_planner" value="true" />
49  </include>
50  <include file="$(find tfg_simulations)/launch/p3_move_base.launch"
      if="$(arg teb_planner)">
51    <arg name="teb_planner" value="true" />
```

```
52   </include>
53
54   <!-- Show everything in Rviz -->
55   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
         tfg_simulations)/config/autonomous_base.rviz"
56     respawn="true" if="$(arg base_planner)"/>
57   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
         tfg_simulations)/config/autonomous_teb.rviz"
58     respawn="true" if="$(arg teb_planner)"/>
59
60 </launch>
```

p3_autonav.launch

```
1 <launch>
2
3   <arg name="base_planner" default="false"/>
4   <arg name="teb_planner" default="false"/>
5
6
7 <!-- Publish scans from best pose at a max of 10 Hz -->
8   <node pkg="amcl" type="amcl" name="amcl" output="screen"
        args="scan:=scan map:=map">
9     <param name="use_map_topic" value="false"/>
10    <param name="base_frame_id" value="base_footprint"/>
11    <param name="odom_model_type" value="diff"/>
12    <param name="odom_alpha5" value="0.1"/>
13    <param name="transform_tolerance" value="0.1" />
14    <param name="gui_publish_rate" value="10.0"/>
15    <param name="laser_max_beams" value="30"/>
16    <param name="min_particles" value="50"/>
17    <param name="max_particles" value="500"/>
18    <param name="kld_err" value="0.05"/>
19    <param name="kld_z" value="0.99"/>
20    <param name="odom_alpha1" value="0.2"/>
21    <param name="odom_alpha2" value="0.2"/>
22    <!-- translation std dev, m -->
23    <param name="odom_alpha3" value="0.8"/>
24    <param name="odom_alpha4" value="0.2"/>
```

```
25      <param name="laser_z_hit" value="0.5"/>
26      <param name="laser_z_short" value="0.05"/>
27      <param name="laser_z_max" value="0.05"/>
28      <param name="laser_z_rand" value="0.5"/>
29      <param name="laser_sigma_hit" value="0.2"/>
30      <param name="laser_lambda_short" value="0.1"/>
31      <param name="laser_lambda_short" value="0.1"/>
32      <param name="laser_model_type" value="likelihood_field"/>
33      <!-- <param name="laser_model_type" value="beam"/> -->
34      <param name="laser_likelihood_max_dist" value="2.0"/>
35      <param name="update_min_d" value="0.2"/>
36      <param name="update_min_a" value="0.5"/>
37      <param name="odom_frame_id" value="odom"/>
38      <param name="resample_interval" value="1"/>
39      <param name="recovery_alpha_slow" value="0.0"/>
40      <param name="recovery_alpha_fast" value="0.0"/>
41    </node>
42
43    <!-- Base local planner -->
44    <node pkg="move_base" type="move_base" respawn="false"
          name="move_base_base" output="screen" if="$(arg base_planner)">
45      <rosparam file="$(find
            tfg_simulations)/config/p3_base/costmap_common_params.yaml"
            command="load" ns="global_costmap" />
46      <rosparam file="$(find
            tfg_simulations)/config/p3_base/costmap_common_params.yaml"
            command="load" ns="local_costmap" />
47      <rosparam file="$(find
            tfg_simulations)/config/p3_base/local_costmap_params.yaml"
            command="load" />
48      <rosparam file="$(find
            tfg_simulations)/config/p3_base/global_costmap_params.yaml"
            command="load" />
49      <rosparam file="$(find
            tfg_simulations)/config/p3_base/local_planner_params.yaml"
            command="load" />
50
51      <param name="controller_frequency" value="10.0"/>
```

```
52      <param name="base_local_planner"
            value="base_local_planner/TrajectoryPlannerROS" />
53    </node>
54
55    <!-- TEB local planner -->
56    <node pkg="move_base" type="move_base" respawn="false"
          name="move_base_teb" output="screen" if="$(arg teb_planner)">
57      <rosparam file="$(find
            tfg_simulations)/config/p3_teb/costmap_common_params.yaml"
            command="load" ns="global_costmap" />
58      <rosparam file="$(find
            tfg_simulations)/config/p3_teb/costmap_common_params.yaml"
            command="load" ns="local_costmap" />
59      <rosparam file="$(find
            tfg_simulations)/config/p3_teb/local_costmap_params.yaml"
            command="load" />
60      <rosparam file="$(find
            tfg_simulations)/config/p3_teb/global_costmap_params.yaml"
            command="load" />
61      <rosparam file="$(find
            tfg_simulations)/config/p3_teb/local_planner_params.yaml"
            command="load" />
62
63      <param name="controller_frequency" value="10.0"/>
64      <param name="base_local_planner"
            value="teb_local_planner/TebLocalPlannerROS" />
65      <param name="base_global_planner" value="navfn/NavfnROS" />
66    </node>
67
68
69  </launch>
```

p3_move_base.launch

```
1  <launch>
2    <!-- these are the arguments you can pass this launch file, for
         example paused:=true -->
3    <arg name="paused" default="false"/>
4    <arg name="use_sim_time" default="true"/>
```

```
5    <arg name="gui" default="false"/>
6    <arg name="headless" default="true"/>
7    <arg name="debug" default="false"/>
8
9    <!--Gazebo parameters -->
10   <include file="$(find gazebo_ros)/launch/empty_world.launch">
11     <arg name="world_name" value="$(find
         tfg_simulations)/worlds/p4_multinav.world"/>
12     <arg name="debug" value="$(arg debug)" />
13     <arg name="gui" value="$(arg gui)" />
14     <arg name="paused" value="$(arg paused)"/>
15     <arg name="use_sim_time" value="$(arg use_sim_time)"/>
16     <arg name="headless" value="$(arg headless)"/>
17   </include>
18
19   <arg name="robot1" value="create_1" />
20   <arg name="x1" value="-5" />
21   <arg name="y1" value="-5" />
22
23   <arg name="robot2" value="create_2" />
24   <arg name="x2" value="-5" />
25   <arg name="y2" value="5" />
26
27   <arg name="robot3" value="create_3" />
28   <arg name="x3" value="5" />
29   <arg name="y3" value="5" />
30
31   <arg name="robot4" value="create_4" />
32   <arg name="x4" value="5" />
33   <arg name="y4" value="-5" />
34
35   <node pkg="map_server" type="map_server" args="$(find
       tfg_simulations)/map/square.yaml" respawn="true" name="map_server"
       >
36     <param name="frame_id" value="map" />
37   </node>
38
39    <!-- Show everything in Rviz -->
```

```
40   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
       tfg_simulations)/config/autonomous_multi_eband.rviz"
41     respawn="true"/>
42
43   <include file="$(find tfg_simulations)/launch/p4_robot.launch" >
44     <arg name="my_ns" value="$(arg robot1)" />
45     <arg name="x_init" value="$(arg x1)" />
46     <arg name="y_init" value="$(arg y1)" />
47   </include>
48
49   <include file="$(find tfg_simulations)/launch/p4_robot.launch" >
50     <arg name="my_ns" value="$(arg robot2)" />
51     <arg name="x_init" value="$(arg x2)" />
52     <arg name="y_init" value="$(arg y2)" />
53   </include>
54
55   <include file="$(find tfg_simulations)/launch/p4_robot.launch" >
56     <arg name="my_ns" value="$(arg robot3)" />
57     <arg name="x_init" value="$(arg x3)" />
58     <arg name="y_init" value="$(arg y3)" />
59   </include>
60
61   <include file="$(find tfg_simulations)/launch/p4_robot.launch" >
62     <arg name="my_ns" value="$(arg robot4)" />
63     <arg name="x_init" value="$(arg x4)" />
64     <arg name="y_init" value="$(arg y4)" />
65   </include>
66
67 </launch>
```

p4_multinav.launch

```
1 <launch>
2
3   <arg name="my_ns" default="create"/>
4   <arg name="x_init" default="0"/>
5   <arg name="y_init" default="0"/>
6
7   <group ns="$(arg my_ns)">
```

```
8
9     <!-- Robot URDF / Gazebo description -->
10    <param name="robot_description" command="$(find xacro)/xacro '$(find
          tfg_simulations)/urdf/create2.xacro'" />
11
12        <!-- Spawn robot model in Gazebo -->
13    <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
          respawn="false" output="screen"
14      args="-urdf -model $(arg my_ns) -x $(arg x_init) -y $(arg y_init)
            -param robot_description"/>
15
16    <!-- Publishes joints state at certain rate -->
17    <node name="joint_state_publisher" pkg="joint_state_publisher"
          type="joint_state_publisher">
18      <param name="use_gui" value="false"/>
19      <param name="rate" value="20"/>
20    </node>
21
22    <!-- Publishes all the tf for the robot -->
23    <node name="robot_state_publisher" pkg="robot_state_publisher"
          type="robot_state_publisher">
24      <param name="tf_prefix" value="$(arg my_ns)" />
25    </node>
26
27    <include file="$(find tfg_simulations)/launch/p4_move_base.launch" >
28      <arg name="my_ns" value="$(arg my_ns)" />
29      <arg name="x_init" value="$(arg x_init)" />
30      <arg name="y_init" value="$(arg y_init)" />
31    </include>
32
33  </group>
34
35 </launch>
```

p4_robot.launch

```
1 <launch>
2
3   <arg name="my_ns" default=""/>
```

```
4   <arg name="x_init" default="0"/>
5   <arg name="y_init" default="0"/>
6
7 <!-- Publish scans from best pose at a max of 10 Hz -->
8   <node pkg="amcl" type="amcl" name="amcl" output="screen"
       args="scan:=/$(arg my_ns)/scan map:=/map">
9     <param name="use_map_topic" value="true"/>
10    <param name="base_frame_id" value="$(arg my_ns)/base_link"/>
11    <param name="odom_model_type" value="diff"/>
12    <param name="odom_alpha5" value="0.1"/>
13    <param name="transform_tolerance" value="0.2" />
14    <param name="gui_publish_rate" value="10.0"/>
15    <param name="laser_max_beams" value="30"/>
16    <param name="min_particles" value="500"/>
17    <param name="max_particles" value="5000"/>
18    <param name="kld_err" value="0.05"/>
19    <param name="kld_z" value="0.99"/>
20    <param name="odom_alpha1" value="0.2"/>
21    <param name="odom_alpha2" value="0.2"/>
22    <!-- translation std dev, m -->
23    <param name="odom_alpha3" value="0.8"/>
24    <param name="odom_alpha4" value="0.2"/>
25    <param name="laser_z_hit" value="0.5"/>
26    <param name="laser_z_short" value="0.05"/>
27    <param name="laser_z_max" value="0.05"/>
28    <param name="laser_z_rand" value="0.5"/>
29    <param name="laser_sigma_hit" value="0.2"/>
30    <param name="laser_lambda_short" value="0.1"/>
31    <param name="laser_lambda_short" value="0.1"/>
32    <param name="laser_model_type" value="likelihood_field"/>
33    <!-- <param name="laser_model_type" value="beam"/> -->
34    <param name="laser_likelihood_max_dist" value="2.0"/>
35    <param name="update_min_d" value="0.2"/>
36    <param name="update_min_a" value="0.5"/>
37    <param name="odom_frame_id" value="$(arg my_ns)/odom"/>
38    <param name="resample_interval" value="1"/>
39    <param name="transform_tolerance" value="0.1"/>
40    <param name="recovery_alpha_slow" value="0.0"/>
41    <param name="recovery_alpha_fast" value="0.0"/>
```

```
42
43      <param name="initial_pose_x" value="$(arg x_init)" />
44      <param name="initial_pose_y" value="$(arg y_init)" />
45
46    </node>
47
48
49    <node pkg="move_base" type="move_base" respawn="false"
          name="move_base" output="screen">
50      <rosparam file="$(find
            tfg_simulations)/config/p5_teb/costmap_common_params.yaml"
            command="load" ns="global_costmap" subst_value="true">
51        <arg name="my_ns" value="$(arg my_ns)" />
52      </rosparam>
53      <rosparam file="$(find
            tfg_simulations)/config/p5_teb/costmap_common_params.yaml"
            command="load" ns="local_costmap" subst_value="true">
54        <arg name="my_ns" value="$(arg my_ns)" />
55      </rosparam>
56      <rosparam file="$(find
            tfg_simulations)/config/p5_teb/local_costmap_params.yaml"
            command="load" subst_value="true">
57        <arg name="my_ns" value="$(arg my_ns)" />
58      </rosparam>
59      <rosparam file="$(find
            tfg_simulations)/config/p5_teb/global_costmap_params.yaml"
            command="load" subst_value="true">
60        <arg name="my_ns" value="$(arg my_ns)" />
61      </rosparam>
62      <rosparam file="$(find
            tfg_simulations)/config/p5_teb/local_planner_params.yaml"
            command="load" subst_value="true"/>
63
64      <param name="controller_frequency" value="4.0"/>
65      <param name="base_local_planner"
            value="teb_local_planner/TebLocalPlannerROS" />
66      <param name="base_global_planner" value="navfn/NavfnROS" />
67
68    </node>
```

```
69
70 </launch>
```

p5_move_base.launch

```
1 <launch>
2   <!-- these are the arguments you can pass this launch file, for
       example paused:=true -->
3   <arg name="paused" default="false"/>
4   <arg name="use_sim_time" default="true"/>
5   <arg name="gui" default="false"/>
6   <arg name="headless" default="true"/>
7   <arg name="debug" default="false"/>
8
9   <!--Gazebo parameters -->
10  <include file="$(find gazebo_ros)/launch/empty_world.launch">
11    <arg name="world_name" value="$(find
         tfg_simulations)/worlds/p4_multinav.world"/>
12    <arg name="debug" value="$(arg debug)" />
13    <arg name="gui" value="$(arg gui)" />
14    <arg name="paused" value="$(arg paused)"/>
15    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
16    <arg name="headless" value="$(arg headless)"/>
17  </include>
18
19  <arg name="robot1" value="create_1" />
20  <arg name="x1" value="-4" />
21  <arg name="y1" value="-5" />
22
23  <arg name="robot2" value="create_2" />
24  <arg name="x2" value="-4" />
25  <arg name="y2" value="-3" />
26
27  <arg name="robot3" value="create_3" />
28  <arg name="x3" value="-4" />
29  <arg name="y3" value="2" />
30
31  <arg name="robot4" value="create_4" />
32  <arg name="x4" value="-4" />
```

```
33   <arg name="y4" value="4.5" />
34
35   <node pkg="map_server" type="map_server" args="$(find
       tfg_simulations)/map/square.yaml" respawn="true" name="map_server"
       >
36     <param name="frame_id" value="map" />
37   </node>
38
39    <!-- Show everything in Rviz -->
40   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
       tfg_simulations)/config/autonomous_multi_teb.rviz"
41     respawn="true"/>
42
43   <include file="$(find tfg_simulations)/launch/p6_robot.launch" >
44     <arg name="my_ns" value="$(arg robot1)" />
45     <arg name="x_init" value="$(arg x1)" />
46     <arg name="y_init" value="$(arg y1)" />
47   </include>
48
49   <include file="$(find tfg_simulations)/launch/p6_robot.launch" >
50     <arg name="my_ns" value="$(arg robot2)" />
51     <arg name="x_init" value="$(arg x2)" />
52     <arg name="y_init" value="$(arg y2)" />
53   </include>
54
55   <include file="$(find tfg_simulations)/launch/p6_robot.launch" >
56     <arg name="my_ns" value="$(arg robot3)" />
57     <arg name="x_init" value="$(arg x3)" />
58     <arg name="y_init" value="$(arg y3)" />
59   </include>
60
61   <include file="$(find tfg_simulations)/launch/p6_robot.launch" >
62     <arg name="my_ns" value="$(arg robot4)" />
63     <arg name="x_init" value="$(arg x4)" />
64     <arg name="y_init" value="$(arg y4)" />
65   </include>
66
67 </launch>
```

p6_multirobot.launch

Carpeta map

```
1 image: map.pgm
2 resolution: 0.05
3 origin: [-100.0, -100.0, 0.0]
4 occupied_thresh: 0.65
5 free_thresh: 0.196
6 negate: 0
```

map.yaml

Carpeta msg

```
1 Task task
2 string robot_id
```

AuctionResult.msg

```
1 float64 bid
```

Bid.msg

```
1 float64 r3x
2 float64 r3y
3 float64 l3x
4 float64 l3y
```

Joystick.msg

```
1 loat64 x1
2 float64 y1
3 float64 x2
4 float64 y2
```

Task.msg

Carpeta scripts

```
1 #!/usr/bin/env python
```

```
2
3  import rospy
4  import sys
5  import time
6  from geometry_msgs.msg import PoseStamped, Twist
7  from actionlib_msgs.msg import GoalStatusArray
8  from tfg_simulations.msg import Task, Bid, AuctionResult
9  from std_msgs.msg import Int16
10
11 robot_ns = 'create_'
12 n_robots = 4
13 auction_time = 1.0 # Seconds
14 max_n_robots_idle = 1
15 real = False
16
17 class Auction():
18     def __init__(self, n_robots, robot_ns, auction_time, real):
19         if real:
20             self.x1 = [1.0, -0.5, 1.0, -0.5, 1.0, -0.5, 1.0, -0.5]
21             self.y1 = [1.5, 0.75, 1.5, 0.75, 1.5, 0.75, 1.5, 0.75]
22             self.x2 = [-0.5, 0.5, -0.5, 0.5, -0.5, 0.5, -0.5, 0.5]
23             self.y2 = [-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0]
24             # self.x1 = [1.0, 0.0, -0.5, 0.25, 1.0, -0.5, 0.25, -0.25]
25             # self.y1 = [1.0, -1.0, 0.0, 0.25, 1.0, -1.0, -0.25, -0.25]
26             # self.x2 = [-0.25, 1.0, 0.0, 0.0, 0.5, -0.5, 0.0, 1.0]
27             # self.y2 = [-0.25, 1.5, 1.5, -1.25, -1.25, 1.25, 1.0, 1.75]
28         else:
29             # self.x1 = [1.0, -2.0, 3.0, 2.0, 1.0, 2.0, -3.0, 2.0]
30             # self.y1 = [-2.0, 3.0, 1.0, -1.0, 4.0, -1.0, 2.0, 1.0]
31             # self.x2 = [-3.0, 4.0, -1.0, -1.0, 1.0, -2.0, 3.0, -2.0]
32             # self.y2 = [4.0, -2.0, 2.0, 3.0, -1.0, 2.0, 3.0, -1.0]
33             self.x1 = [-3.0, -3.0, -3.0, -3.0, -3.0, -3.0, -3.0, -3.0]
34             self.y1 = [5.0, -5.0, 2.0, -2.0, 5.0, -5.0, 2.0, -2.0]
35             self.x2 = [3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0]
36             self.y2 = [2.0, -2.0, 5.0, -5.0, 2.0, -2.0, 5.0, -5.0]
37         self.n_tasks = len(self.x1)
38         self.done_tasks = 0
39         self.n_robots = n_robots
40         self.bids = [-1]*n_robots
```

```
41        self.robot_ns = robot_ns
42        self.states = [-1]*n_robots
43        self.auction_time = auction_time
44        self.auction_pub = rospy.Publisher('auction', Task, queue_size=1)
45        self.result_pub = rospy.Publisher('auction_result',
              AuctionResult, queue_size=1)
46
47        for i in range(1, self.n_robots + 1):
48            rospy.Subscriber(str(robot_ns) + str(i) + '/bid', Bid,
                  self.bid_callback, i-1)
49
50        for i in range(1, self.n_robots + 1):
51            rospy.Subscriber(str(robot_ns) + str(i) + '/bidder_ready',
52                                    Int16, self.state_callback, i-1)
53
54    def bid_callback(self, msg, i):
55        self.bids[i] = msg.bid
56
57    def state_callback(self, msg, i):
58        self.states[i] = msg.data
59
60    def start_auction(self):
61        for i in range(1, self.n_robots + 1):
62            if auction.done_tasks < auction.n_tasks:
63                self.bids = [-1]*n_robots
64                task = Task()
65                task.x1 = self.x1[self.done_tasks]
66                task.y1 = self.y1[self.done_tasks]
67                task.x2 = self.x2[self.done_tasks]
68                task.y2 = self.y2[self.done_tasks]
69                self.auction_pub.publish(task)
70                time.sleep(self.auction_time)
71
72                print(self.bids)
73
74                min_bid = 1e6
75                min_bid_id = -1
76                for j in range(1, self.n_robots + 1):
77                    bid = self.bids[j-1]
```

```
78                  if bid == -1: # If robot has not bid
79                      print(str(self.robot_ns) + str(j) + ' has not bid')
80                  else:
81                      if bid < min_bid:
82                          min_bid = bid
83                          min_bid_id = j
84
85              if min_bid_id != -1:
86                  print(min_bid)
87                  result = AuctionResult()
88                  result.task = task
89                  result.robot_id = str(self.robot_ns) + str(min_bid_id)
90                  self.result_pub.publish(result)
91                  self.done_tasks += 1
92
93              time.sleep(0.2)
94
95          print('Auction finished')
96          print('')
97
98
99  if __name__ == '__main__':
100     rospy.init_node('auctioneer')
101     rate = rospy.Rate(5.0)
102
103     print('Auctioneer ready')
104     print('')
105
106     if len(sys.argv) > 1:
107         robot_ns = sys.argv[1]
108
109     if len(sys.argv) > 2:
110         n_robots = int(sys.argv[2])
111
112     auction = Auction(n_robots, robot_ns, auction_time, real)
113     finished = False
114
115     while (not rospy.is_shutdown()) and (not finished):
116         if auction.done_tasks < auction.n_tasks:
```

```
            if auction.states.count(1) > max_n_robots_idle:
                if auction.done_tasks == 0:
                    t_init = rospy.get_time()
                auction.start_auction()

        else:
            if auction.states == [1]*n_robots:
                finished = True
                t_end = rospy.get_time()
                print('')
                print('Total time: ' + str(t_end-t_init) + ' seconds')

        rate.sleep()
```

auctioneer.py

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

import csv
import numpy as np
import rospy
import sys
import time
import tf
from geometry_msgs.msg import PoseStamped, Twist
from actionlib_msgs.msg import GoalStatusArray

my_ns = "create" # Default name
destinations = [[-1.5, 6.5, 3.14], [-3.0, -0.5, 3.14], [-3.0, 3.0, 2.0],
                  [-4.0, -4.0, 5.0], [0.0, 0.0, 0.0]]

vel_array = []
time_array = []


def callback(msg):
    global status
    if len(msg.status_list) == 0:
        status = -1
```

```
24      else:
25          status = msg.status_list[-1].status
26
27  def callback2(msg):
28      global vel
29      vel = msg.linear.x
30
31  if __name__ == '__main__':
32      global status
33      global vel
34
35      rospy.init_node('goal_publisher')
36      pub = rospy.Publisher('move_base_simple/goal', PoseStamped,
            queue_size=1)
37      rospy.Subscriber('/move_base/status', GoalStatusArray, callback)
38      rospy.Subscriber('/cmd_vel', Twist, callback2)
39      rate = rospy.Rate(10.0)
40
41      goal = PoseStamped()
42      goal.header.frame_id = "map"
43      goal.header.stamp = rospy.Time.now()
44      goal.pose.position.x = 0
45      goal.pose.position.y = 0
46      goal.pose.position.z = 0
47      goal.pose.orientation.x = 0
48      goal.pose.orientation.y = 0
49      goal.pose.orientation.z = 0
50      goal.pose.orientation.w = 1
51      time.sleep(3.0)
52      pub.publish(goal)
53      print('Published')
54
55      while status != 3: # Ready to start
56          pass
57
58      iter = 0 # Loop counter
59      n_dest = 0 # Number of next destination
60      while not rospy.is_shutdown():
61          if status == 3: # Publish goal
```

```
62            if int(n_dest/len(destinations)) == 1:
63                n_dest = 0
64                iter += 1
65            i = 0
66            vel_array = []
67            time_array = []
68            goal.header.stamp = rospy.Time.now()
69            quaternion = tf.transformations.quaternion_from_euler(0, 0,
                  destinations[n_dest][2])
70            goal.pose.position.x = destinations[n_dest][0]
71            goal.pose.position.y = destinations[n_dest][1]
72            goal.pose.position.z = 0
73            goal.pose.orientation.x = quaternion[0]
74            goal.pose.orientation.y = quaternion[1]
75            goal.pose.orientation.z = quaternion[2]
76            goal.pose.orientation.w = quaternion[3]
77            pub.publish(goal)
78            n_dest += 1
79            time.sleep(2.0)
80            t1 = rospy.get_time()
81
82
83        t2 = rospy.get_time()
84        time_array.append(t2-t1)
85        t1 = t2
86        vel_array.append(np.abs(vel))
87
88
89        i += 1
90        rate.sleep()
91
92        # Print statistics of experiment
93        if status == 3:
94            print('')
95            print('Iteration # ' + str(iter+1))
96            print('Destination # ' + str(n_dest))
97            time_array = np.array(time_array)
98            vel_array = np.array(vel_array)
99            total_time = np.sum(time_array)
```

```
100            mean_vel = np.mean(vel_array)
101            print('Average velocity: ' + str(mean_vel))
102            print('Total time: ' + str(total_time))
103            print('Estimated distance: ' + str(mean_vel*total_time))
```

benchmark.py

```
1 #!/usr/bin/env python
2
3 import rospy
4 import sys
5 import time
6 import math
7 import rosnode
8 from geometry_msgs.msg import Pose, PoseStamped,
      PoseWithCovarianceStamped, Twist
9 from actionlib_msgs.msg import GoalStatusArray
10 from tfg_simulations.msg import Task, Bid, AuctionResult
11 from std_msgs.msg import Int16
12
13 wait_time_check_state = 2.0
14
15 class Robot():
16     def __init__(self, robot_ns):
17         self.robot_ns = robot_ns
18         self.pose = Pose()
19         self.velocity = 0.4
20         self.state = -1
21         self.ready = 0
22
23         self.bid_pub = rospy.Publisher('bid', Bid, queue_size=1)
24         self.move_pub = rospy.Publisher('move_base_simple/goal',
               PoseStamped, queue_size=1)
25         self.ready_pub = rospy.Publisher('bidder_ready', Int16,
               queue_size=1)
26
27         rospy.Subscriber('amcl_pose', PoseWithCovarianceStamped,
               self.pose_callback)
28         rospy.Subscriber('/auction', Task, self.task_callback)
```

```
29          rospy.Subscriber('/auction_result', AuctionResult,
                self.result_callback)
30          rospy.Subscriber('move_base/status', GoalStatusArray,
                self.state_callback)
31
32
33      def pose_callback(self, msg):
34          self.pose = msg.pose.pose
35
36      def task_callback(self, msg):
37          if self.ready:
38              distance = (math.sqrt((msg.x2-msg.x1)**2+(msg.y2-msg.y1)**2)
                    +
39                          math.sqrt((msg.x1-self.pose.position.x)**2+(msg.y1-self.pose.posit
40
41              self.bid_pub.publish(distance/self.velocity)
42
43      def result_callback(self, msg):
44          if msg.robot_id == self.robot_ns:
45              self.ready = 0
46              self.ready_pub.publish(0) # Not ready
47              goal = self.calculate_goal(msg.task.x1, msg.task.y1)
48              self.move_pub.publish(goal)
49              time.sleep(wait_time_check_state)
50              while self.state != 3:
51                  time.sleep(0.2)
52
53              goal = self.calculate_goal(msg.task.x2, msg.task.y2)
54              self.move_pub.publish(goal)
55              time.sleep(wait_time_check_state)
56              while self.state != 3:
57                  time.sleep(0.2)
58
59              self.ready = 1
60              self.ready_pub.publish(1) # Ready again
61
62      def state_callback(self, msg):
63          if len(msg.status_list) == 0:
64              self.state = -1
```

```
65          else:
66              self.state = msg.status_list[-1].status
67
68      def calculate_goal(self, x, y):
69          goal = PoseStamped()
70          goal.header.frame_id = "map"
71          goal.header.stamp = rospy.Time.now()
72          goal.pose.position.x = x
73          goal.pose.position.y = y
74          goal.pose.position.z = 0
75          goal.pose.orientation.x = 0
76          goal.pose.orientation.y = 0
77          goal.pose.orientation.z = 0
78          goal.pose.orientation.w = 1
79          return goal
80
81
82
83  if __name__ == '__main__':
84      rospy.init_node('bidder')
85      rate = rospy.Rate(2.0)
86
87      robot_ns = sys.argv[1]
88      robot = Robot(robot_ns)
89      started = False
90
91      while not rospy.is_shutdown():
92          if not started:
93              nodes = rosnode.get_node_names()
94
95              if '/auctioneer' in nodes:
96                  started = True
97                  robot.velocity =
                        rospy.get_param('move_base/TebLocalPlannerROS/max_vel_x')
98                  robot.ready_pub.publish(1)
99                  robot.ready = 1
100
101         if robot.ready:
102             robot.ready_pub.publish(1)
```

```
103         rate.sleep()
```

bidder.py

```
1 #!/usr/bin/env python3
2 
3 import rospy
4 import tfg_simulations.msg
5 from geometry_msgs.msg import Twist
6 
7 vel_max = 0.5 # Max Create linear velocity
8 w_max = 4.25 # Max Create angular velocity
9 vel = 0
10 w = 0
11 
12 def callback(msg):
13     global vel, w
14     vel = msg.r3y * (-1) * vel_max
15     w = msg.l3x * (-1) * w_max
16 
17 
18 if __name__ == '__main__':
19     rospy.init_node('create_cmd')
20     pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
21     rate = rospy.Rate(10.0)
22 
23     cmd = Twist()
24     cmd.linear.x = 0
25     cmd.linear.y = 0
26     cmd.linear.z = 0
27     cmd.angular.x = 0
28     cmd.angular.y = 0
29     cmd.angular.z = 0
30 
31     rospy.Subscriber('axis', tfg_simulations.msg.Joystick, callback)
32 
33     while not rospy.is_shutdown():
34         cmd.linear.x = vel
35         cmd.angular.z = w
```

```
36            pub.publish(cmd)
37            rate.sleep()
```

cmd_create.py

```
1  #!/usr/bin/env python
2
3  import rospy
4  import sys
5  import tf
6  from geometry_msgs.msg import PoseStamped
7
8  my_ns = "create" # Default name
9
10 if __name__ == '__main__':
11
12     my_ns = sys.argv[1]
13     rospy.init_node('goal_publisher')
14     if my_ns == "create":
15         print("\nStarted process with default namespace: " + my_ns)
16     else:
17         print("\nStarted process with namespace: " + my_ns)
18
19
20     pub = rospy.Publisher('move_base_simple/goal', PoseStamped,
           queue_size=1)
21     goal = PoseStamped()
22
23     while not rospy.is_shutdown():
24         x = float(input("X position: "))
25         y = float(input("Y position: "))
26         yaw = float(input("Yaw orientation: "))
27
28         goal.header.frame_id = "map"
29         goal.header.stamp = rospy.Time.now()
30
31         quaternion = tf.transformations.quaternion_from_euler(0, 0, yaw)
32         goal.pose.position.x = x
33         goal.pose.position.y = y
```

```
34        goal.pose.position.z = 0
35        goal.pose.orientation.x = quaternion[0]
36        goal.pose.orientation.y = quaternion[1]
37        goal.pose.orientation.z = quaternion[2]
38        goal.pose.orientation.w = quaternion[3]
39
40        pub.publish(goal)
```

goal_pub.py

```
1 #!/usr/bin/env python3
2
3 import rospy
4 import pygame
5 import sys
6 from geometry_msgs.msg import Twist
7
8 pygame.init()
9 pygame.display.set_mode((200, 200))
10 print(sys.version)
11
12 vel_max = 0.5
13 w_max = 4.25
14 vel = 0
15 w = 0
16
17 if __name__ == '__main__':
18     rospy.init_node('keyboard')
19     pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
20     rate = rospy.Rate(100.0)
21
22     rospy.Time()
23
24     cmd = Twist()
25     cmd.linear.x = 0
26     cmd.linear.y = 0
27     cmd.linear.z = 0
28     cmd.angular.x = 0
29     cmd.angular.y = 0
```

```
30    cmd.angular.z = 0
31
32    while not rospy.is_shutdown():
33        for event in pygame.event.get():
34            if event.type == pygame.KEYDOWN:
35                if event.key == pygame.K_UP:
36                    cmd.linear.x = cmd.linear.x + vel_max
37                if event.key == pygame.K_DOWN:
38                    cmd.linear.x = cmd.linear.x - vel_max
39                if event.key == pygame.K_LEFT:
40                    cmd.angular.z = cmd.angular.z + w_max
41                if event.key == pygame.K_RIGHT:
42                    cmd.angular.z = cmd.angular.z - w_max
43
44            if event.type == pygame.KEYUP:
45                if event.key == pygame.K_UP:
46                    cmd.linear.x = cmd.linear.x - vel_max
47                if event.key == pygame.K_DOWN:
48                    cmd.linear.x = cmd.linear.x + vel_max
49                if event.key == pygame.K_LEFT:
50                    cmd.angular.z = cmd.angular.z - w_max
51                if event.key == pygame.K_RIGHT:
52                    cmd.angular.z = cmd.angular.z + w_max
53
54        pub.publish(cmd)
55        rate.sleep()
```

keyboard_pub.py

```
1 #!/usr/bin/env python3
2 import rospy
3 import tfg_simulations.msg
4 import pygame
5
6 pygame.init()
7 pygame.joystick.init()
8 joystick = pygame.joystick.Joystick(0)
9 joystick.init()
10
```

```
11 print ('Initialized Joystick : %s' % joystick.get_name())
12 # pygame.display.set_mode((200, 200))
13
14 if __name__ == '__main__':
15     rospy.init_node('joystick_controller')
16     pub = rospy.Publisher('axis', tfg_simulations.msg.Joystick,
           queue_size=1)
17     rate = rospy.Rate(50.0)
18
19     numaxes = joystick.get_numaxes()
20     print("numaxes")
21     print(numaxes)
22     print("---------------")
23     numbuttons = joystick.get_numbuttons()
24     print("numbuttons")
25     print(numbuttons)
26     print("---------------")
27
28     while not rospy.is_shutdown():
29         cmd = tfg_simulations.msg.Joystick()
30         pygame.event.pump()
31         cmd.l3x = joystick.get_axis(0)
32         cmd.l3y = joystick.get_axis(1)
33         cmd.r3x = joystick.get_axis(3)
34         cmd.r3y = joystick.get_axis(4)
35
36         pub.publish(cmd)
37         rate.sleep()
```

ps3_node.py

Carpeta urdf

```
1 <?xml version="1.0"?>
2 <!-- iRobot Roomba 650 model -->
3 <robot name="roomba" xmlns:xacro="http://www.ros.org/wiki/xacro">
4
5 <!-- Robot form parameters -->
6   <xacro:property name="PI" value="3.141592"/>
```

```
7   <xacro:property name="wheels_offset" value="0.235"/>
8   <xacro:property name="wheels_xgap" value="0.0"/>
9   <xacro:property name="wheels_radius" value="0.036"/>
10  <xacro:property name="wheels_width" value="0.015"/>
11  <xacro:property name="wheels_mass" value="0.7"/>
12  <xacro:property name="wheels_effort" value="1"/>
13  <xacro:property name="wheels_velocity" value="1"/>
14  <xacro:property name="wheels_damping" value="0.1"/>
15  <xacro:property name="base_zgap" value="0.01"/>
16  <xacro:property name="base_height" value="${0.083-base_zgap}"/>
17  <xacro:property name="base_radius" value="0.174"/>
18  <xacro:property name="base_mass" value="3.0"/>
19  <xacro:property name="roller_radius" value="${base_zgap*1.0}"/>
20  <xacro:property name="roller_xgap" value="0.135"/>
21  <xacro:property name="ir_height" value="0.009"/>
22  <xacro:property name="ir_radius" value="0.01"/>
23  <xacro:property name="ir_mass" value="0.01"/>
24  <xacro:property name="ir_xgap" value="0.15"/>
25  <xacro:property name="lidar_radius" value="0.038"/>
26  <xacro:property name="lidar_height" value="0.041"/>
27  <!-- <xacro:property name="lidar_offset_y" value="0.105"/> -->
28  <xacro:property name="lidar_offset_y" value="0.0"/>
29  <xacro:property name="local_costmap_offset_x" value="2.0"/>
30  <xacro:property name="local_costmap_offset_y" value="0.0"/>
31
32
33  <xacro:macro name="cylinder_inertial" params="radius length mass">
34    <inertial>
35      <origin xyz="0 0 0" rpy="0 0 0"/>
36      <mass value="${mass}"/>
37      <inertia ixx="${mass*radius*radius/2}" ixy="0.0" ixz="0.0"
38              iyy="${mass*radius*radius/4+mass*length*length/12}"
                    iyz="0.0"
39              izz="${mass*radius*radius/4+mass*length*length/12}"/>
40    </inertial>
41  </xacro:macro>
42
43  <xacro:include filename="$(find tfg_simulations)/urdf/create2.gazebo"
      />
```

```
44  <xacro:include filename="$(find tfg_simulations)/urdf/materials.xacro"
       />
45
46  <link name="base_footprint">
47    <inertial>
48      <origin xyz="0 0 0" rpy="0 0 0"/>
49      <mass value="0.01"/>
50      <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
51               iyy="0.0001" iyz="0.0" izz="0.0001"/>
52    </inertial>
53    <visual>
54      <origin xyz="0 0 0" rpy="0 0 0"/>
55      <geometry>
56        <mesh
            filename="package://tfg_simulations/meshes/base_create2.dae"
            />
57      </geometry>
58    </visual>
59    <collision>
60      <origin xyz="0 0 0.05" rpy="0 0 0"/>
61      <geometry>
62        <box size="0.001 0.001 0.001" />
63      </geometry>
64    </collision>
65  </link>
66
67  <joint name="base_footprint_joint" type="fixed">
68    <origin xyz="0 0 ${(base_height/2+base_zgap)}" rpy="0 0 0"/>
69    <parent link="base_footprint"/>
70    <child link="base_link"/>
71  </joint>
72
73  <link name="base_link">
74    <xacro:cylinder_inertial
75      radius="${base_radius}" length="${base_height}"
          mass="${base_mass}"/>
76    <collision>
77      <origin xyz="0 0 0" rpy="0 0 0"/>
78      <geometry>
```

```
79          <cylinder radius="${base_radius}" length="${base_height}"/>
80        </geometry>
81      </collision>
82    </link>
83
84    <joint name="lidar_joint" type="fixed">
85      <origin xyz="0 ${lidar_offset_y} ${base_height/2+lidar_height/2}"
86              rpy="0 0 0"/>
87      <parent link="base_link"/>
88      <child link="lidar"/>
89    </joint>
90
91    <link name="lidar">
92      <inertial>
93        <origin xyz="0 0 0" rpy="0 0 0"/>
94        <mass value="0.01"/>
95        <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
96                 iyy="0.0001" iyz="0.0" izz="0.0001"/>
97      </inertial>
98      <visual>
99        <origin xyz="0 0 ${-lidar_height/2}" rpy="0 0 0"/>
100       <geometry>
101         <!-- <cylinder radius="${lidar_radius}"
               length="${lidar_height}"/> -->
102         <mesh
               filename="package://tfg_simulations/meshes/rplidar_a3m1.dae"
               />
103       </geometry>
104       <material name="purple"/>
105     </visual>
106     <collision>
107       <origin xyz="0 0 0" rpy="0 0 0"/>
108       <geometry>
109         <cylinder radius="${lidar_radius}" length="${lidar_height}"/>
110       </geometry>
111     </collision>
112   </link>
113
114   <joint name="roller_joint" type="fixed">
```

```
115     <origin xyz="${roller_xgap} 0 ${-base_height/2}" rpy="0 0 0"/>
116     <parent link="base_link"/>
117     <child link="roller"/>
118   </joint>
119
120   <link name="roller">
121     <inertial>
122       <origin xyz="0 0 0" rpy="0 0 0"/>
123       <mass value="0.05"/>
124       <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
125                iyy="0.0001" iyz="0.0" izz="0.0001"/>
126     </inertial>
127     <!-- <visual>
128       <origin xyz="0 0 0" rpy="0 0 0"/>
129       <geometry>
130         <sphere radius="${roller_radius}"/>
131       </geometry>
132       <material name="green_transparent"/>
133     </visual> -->
134     <collision>
135       <origin xyz="0 0 0" rpy="0 0 0"/>
136       <geometry>
137         <sphere radius="${roller_radius}"/>
138       </geometry>
139     </collision>
140   </link>
141
142   <!-- <joint name="roller_joint_back" type="fixed">
143     <origin xyz="${-roller_xgap} 0 ${-base_height/2}" rpy="0 0 0"/>
144     <parent link="base_link"/>
145     <child link="roller_back"/>
146   </joint>
147
148   <link name="roller_back">
149     <inertial>
150       <origin xyz="0 0 0" rpy="0 0 0"/>
151       <mass value="0.05"/>
152       <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
153                iyy="0.0001" iyz="0.0" izz="0.0001"/>
```

```
154     </inertial>
155     <visual>
156       <origin xyz="0 0 0" rpy="0 0 0"/>
157       <geometry>
158         <sphere radius="${roller_radius}"/>
159       </geometry>
160       <material name="green_transparent"/>
161     </visual>
162     <collision>
163       <origin xyz="0 0 0" rpy="0 0 0"/>
164       <geometry>
165         <sphere radius="${roller_radius}"/>
166       </geometry>
167     </collision>
168   </link> -->
169
170   <joint name="ir_joint" type="fixed">
171     <origin xyz="${ir_xgap} 0 ${base_height/2+ir_height/2}" rpy="0 0 0"/>
172     <parent link="base_link"/>
173     <child link="ir"/>
174   </joint>
175
176   <link name="ir">
177     <inertial>
178       <origin xyz="0 0 0" rpy="0 0 0"/>
179       <mass value="${ir_mass}"/>
180       <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
181                iyy="0.0001" iyz="0.0" izz="0.0001"/>
182     </inertial>
183     <!-- <visual>
184       <origin xyz="0 0 0" rpy="0 0 0"/>
185       <geometry>
186         <cylinder radius="${ir_radius}" length="${ir_height}"/>
187       </geometry>
188       <material name="red"/>
189     </visual> -->
190     <collision>
191       <origin xyz="0 0 0" rpy="0 0 0"/>
192       <geometry>
```

```
193          <cylinder radius="${ir_radius}" length="${ir_height}"/>
194        </geometry>
195      </collision>
196    </link>
197
198    <link name="right_wheel">
199      <inertial>
200        <origin xyz="0 0 0" rpy="0 0 0"/>
201        <mass value="${wheels_mass}"/>
202        <inertia ixx="0.001" ixy="0.0" ixz="0.0"
203                 iyy="0.001" iyz="0.0" izz="0.001"/>
204      </inertial>
205      <visual>
206        <origin xyz="0 0 0" rpy="0 0 0"/>
207        <geometry>
208          <cylinder radius="${wheels_radius}" length="${wheels_width}"/>
209        </geometry>
210        <material name="black"/>
211      </visual>
212      <collision>
213        <origin xyz="0 0 0" rpy="0 0 0"/>
214        <geometry>
215          <cylinder radius="${wheels_radius}" length="${wheels_width}"/>
216        </geometry>
217      </collision>
218    </link>
219
220    <joint name="right_wheel_joint" type="continuous">
221      <origin xyz="${-wheels_xgap} ${-wheels_offset/2}
           ${-base_zgap-base_height/2+wheels_radius}" rpy="${-PI/2} 0 0"/>
222      <axis xyz="0 0 1"/>
223      <parent link="base_link"/>
224      <child link="right_wheel"/>
225      <dynamics damping="${wheels_damping}"/>
226      <!-- <limit effort="${wheels_effort}" velocity="${wheels_velocity}"
           /> -->
227    </joint>
228
229    <link name="left_wheel">
```

```
230     <inertial>
231       <origin xyz="0 0 0" rpy="0 0 0"/>
232       <mass value="0.25"/>
233       <inertia ixx="0.001" ixy="0.0" ixz="0.0"
234                iyy="0.001" iyz="0.0" izz="0.001"/>
235     </inertial>
236     <visual>
237       <origin xyz="0 0 0" rpy="0 0 0"/>
238       <geometry>
239         <cylinder radius="${wheels_radius}" length="${wheels_width}"/>
240       </geometry>
241       <material name="black"/>
242     </visual>
243     <collision>
244       <origin xyz="0 0 0" rpy="0 0 0"/>
245       <geometry>
246         <cylinder radius="${wheels_radius}" length="${wheels_width}"/>
247       </geometry>
248     </collision>
249   </link>
250
251   <joint name="left_wheel_joint" type="continuous">
252     <origin xyz="${-wheels_xgap} ${wheels_offset/2}
          ${-base_zgap-base_height/2+wheels_radius}" rpy="${-PI/2} 0 0"/>
253     <axis xyz="0 0 1"/>
254     <parent link="base_link"/>
255     <child link="left_wheel"/>
256     <dynamics damping="${wheels_damping}"/>
257     <!-- <limit effort="${wheels_effort}" velocity="${wheels_velocity}"
          /> -->
258   </joint>
259 </robot>
```

create2.xacro

```
1 <?xml version="1.0"?>
2 <robot>
3
4   <gazebo>
```

```
5    <plugin name="differential_drive_controller"
       filename="libgazebo_ros_diff_drive.so">
6      <alwaysOn>true</alwaysOn>
7      <updateRate>50</updateRate>
8      <leftJoint>left_wheel_joint</leftJoint>
9      <rightJoint>right_wheel_joint</rightJoint>
10     <wheelSeparation>0.235</wheelSeparation>
11     <wheelDiameter>0.072</wheelDiameter>
12     <wheelTorque>20</wheelTorque>
13     <wheelAcceleration>5</wheelAcceleration>
14     <commandTopic>cmd_vel</commandTopic>
15     <odometryTopic>odom</odometryTopic>
16     <odometryFrame>odom</odometryFrame>
17     <robotBaseFrame>base_footprint</robotBaseFrame>
18   </plugin>
19 </gazebo>
20
21
22 <gazebo reference="base_footprint">
23   <mu1>0</mu1>
24   <mu2>0</mu2>
25 </gazebo>
26
27 <gazebo reference="base_link">
28   <mu1>0</mu1>
29   <mu2>0</mu2>
30 </gazebo>
31
32 <gazebo reference="right_wheel">
33   <mu1>0.8</mu1>
34   <mu2>0.8</mu2>
35   <material>Gazebo/Black</material>
36 </gazebo>
37
38 <gazebo reference="left_wheel">
39   <mu1>0.8</mu1>
40   <mu2>0.8</mu2>
41   <material>Gazebo/Black</material>
42 </gazebo>
```

```
43
44 <gazebo reference="roller">
45   <mu1>0</mu1>
46   <mu2>0</mu2>
47   <material>Gazebo/Green</material>
48 </gazebo>
49
50 <!-- <gazebo reference="roller_back">
51   <mu1>0</mu1>
52   <mu2>0</mu2>
53   <material>Gazebo/Green</material>
54 </gazebo> -->
55
56 <gazebo reference="lidar">
57   <mu1>0.5</mu1>
58   <mu2>0.5</mu2>
59   <sensor type="ray" name="laser_base">
60     <pose>0 0 0 0 0 0</pose>
61     <visualize>false</visualize>
62     <update_rate>40</update_rate>
63     <ray>
64       <scan>
65         <horizontal>
66           <samples>720</samples>
67           <resolution>1</resolution>
68           <min_angle>-3.141592</min_angle>
69           <max_angle>3.141592</max_angle>
70         </horizontal>
71       </scan>
72       <range>
73         <min>0.10</min>
74         <max>30.0</max>
75         <resolution>0.01</resolution>
76       </range>
77       <noise>
78         <type>gaussian</type>
79         <mean>0.0</mean>
80         <stddev>0.01</stddev>
81       </noise>
```

```
82        </ray>
83        <plugin name="gazebo_ros_head_hokuyo_controller"
84                filename="libgazebo_ros_laser.so">
85          <topicName>scan</topicName>
86          <frameName>base_footprint</frameName>
87        </plugin>
88      </sensor>
89    </gazebo>
90
91    <gazebo reference="ir">
92      <mu1>0</mu1>
93      <mu2>0</mu2>
94      <material>Gazebo/Red</material>
95    </gazebo>
96
97  </robot>
```

create2.gazebo

```
1  <?xml version="1.0"?>
2  <robot>
3
4    <material name="black">
5      <color rgba="0.0 0.0 0.0 1.0"/>
6    </material>
7
8    <material name="blue">
9      <color rgba="0.0 0.0 0.8 1.0"/>
10   </material>
11
12   <material name="green">
13     <color rgba="0.0 0.8 0.0 1.0"/>
14   </material>
15
16   <material name="green_transparent">
17     <color rgba="0.0 0.8 0.0 0.75"/>
18   </material>
19
20   <material name="grey">
```

```
21      <color rgba="0.2 0.2 0.2 1.0"/>
22    </material>
23
24    <material name="orange">
25      <color rgba="${255/255} ${108/255} ${10/255} 1.0"/>
26    </material>
27
28    <material name="orange_transparent">
29      <color rgba="${255/255} ${108/255} ${10/255} 0.75"/>
30    </material>
31
32    <material name="brown">
33      <color rgba="${222/255} ${207/255} ${195/255} 1.0"/>
34    </material>
35
36    <material name="red">
37      <color rgba="0.8 0.0 0.0 1.0"/>
38    </material>
39
40    <material name="white">
41      <color rgba="1.0 1.0 1.0 1.0"/>
42    </material>
43
44    <material name="purple">
45      <color rgba="${143/255} ${62/255} ${203/255} 1.0"/>
46    </material>
47
48  </robot>
```

materials.xacro

C.2. Paquete tfg_roomba

Carpeta launch

```
1  <launch>
2
3    <!-- Show in Rviz -->
```

```
4   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
        tfg_roomba)/config/rviz_remote.rviz"/>
5
6   <!-- Joystick commands -->
7   <node name="joystick_publisher" pkg="tfg_roomba" type="ps3_node.py"
        respawn="true"/>
8   <node name="joystick_to_cmd" pkg="tfg_roomba" type="roomba_cmd.py"/>
9
10  <!-- SLAM node -->
11  <node name="mapping" pkg="gmapping" type="slam_gmapping"
        args="scan:=/scan">
12    <param name="base_frame" value="base_link"/>
13    <param name="angularUpdate" value="0.1"/>
14    <param name="delta" value="0.05"/>
15  </node>
16
17 </launch>
```

m1_slam.launch

```
1 <launch>
2
3   <!-- Select planner (ONLY ONE MUST BE TRUE) -->
4   <arg name="base_planner" value="false"/>
5   <arg name="teb_planner" value="true"/>
6   <arg name="dwa_planner" value="false"/>
7
8   <!-- Publishes all the tf for the robot -->
9   <node name="robot_state_publisher" pkg="robot_state_publisher"
        type="robot_state_publisher"
10    respawn="true"/>
11
12  <!-- Spawns a map of the environment -->
13  <node pkg="map_server" type="map_server" args="$(find
        tfg_roomba)/map/home_multi.yaml"
14    respawn="true" name="map_server" />
15
16  <!-- Import move_base file -->
```

```
17   <include file="$(find tfg_roomba)/launch/move_base.launch" if="$(arg
       base_planner)">
18     <arg name="base_planner" value="true" />
19   </include>
20   <include file="$(find tfg_roomba)/launch/move_base.launch" if="$(arg
       teb_planner)">
21     <arg name="teb_planner" value="true" />
22   </include>
23   <include file="$(find tfg_roomba)/launch/move_base.launch" if="$(arg
       dwa_planner)">
24     <arg name="dwa_planner" value="true" />
25   </include>
26
27   <!-- Show everything in Rviz -->
28   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
       tfg_simulations)/config/autonomous_base.rviz"
29     respawn="true" if="$(arg base_planner)"/>
30   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
       tfg_simulations)/config/autonomous_teb.rviz"
31     respawn="true" if="$(arg teb_planner)"/>
32   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
       tfg_simulations)/config/autonomous_dwa.rviz"
33     respawn="true" if="$(arg dwa_planner)"/>
34
35 </launch>
```

m2_autonomous.launch

```
1 <launch>
2
3   <arg name="base_planner" default="false"/>
4   <arg name="teb_planner" default="false"/>
5
6 <!-- Publish scans from best pose at a max of 10 Hz -->
7   <node pkg="amcl" type="amcl" name="amcl" output="screen"
       args="scan:=scan map:=map">
8     <param name="use_map_topic" value="false"/>
9     <param name="base_frame_id" value="base_footprint"/>
10    <param name="odom_model_type" value="diff"/>
```

```
11      <param name="odom_alpha5" value="0.1"/>
12      <param name="transform_tolerance" value="0.3" />
13      <param name="gui_publish_rate" value="10.0"/>
14      <param name="laser_max_beams" value="30"/>
15      <param name="min_particles" value="50"/>
16      <param name="max_particles" value="500"/>
17      <param name="kld_err" value="0.05"/>
18      <param name="kld_z" value="0.99"/>
19      <param name="odom_alpha1" value="0.2"/>
20      <param name="odom_alpha2" value="0.2"/>
21      <!-- translation std dev, m -->
22      <param name="odom_alpha3" value="0.8"/>
23      <param name="odom_alpha4" value="0.2"/>
24      <param name="laser_z_hit" value="0.5"/>
25      <param name="laser_z_short" value="0.05"/>
26      <param name="laser_z_max" value="0.05"/>
27      <param name="laser_z_rand" value="0.5"/>
28      <param name="laser_sigma_hit" value="0.2"/>
29      <param name="laser_lambda_short" value="0.1"/>
30      <param name="laser_lambda_short" value="0.1"/>
31      <param name="laser_model_type" value="likelihood_field"/>
32      <!-- <param name="laser_model_type" value="beam"/> -->
33      <param name="laser_likelihood_max_dist" value="2.0"/>
34      <param name="update_min_d" value="0.2"/>
35      <param name="update_min_a" value="0.5"/>
36      <param name="odom_frame_id" value="odom"/>
37      <param name="resample_interval" value="1"/>
38      <param name="recovery_alpha_slow" value="0.0"/>
39      <param name="recovery_alpha_fast" value="0.0"/>
40      <param name="first_map_only" value="true"/>
41    </node>
42
43    <!-- Base local planner -->
44    <node pkg="move_base" type="move_base" respawn="false"
          name="move_base_base" output="screen" if="$(arg base_planner)">
45      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_base/costmap_common_params.yaml"
            command="load" ns="global_costmap" />
```

```
46      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_base/costmap_common_params.yaml"
            command="load" ns="local_costmap" />
47      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_base/local_costmap_params.yaml"
            command="load" />
48      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_base/global_costmap_params.yaml"
            command="load" />
49      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_base/local_planner_params.yaml"
            command="load" />
50
51      <param name="controller_frequency" value="10.0"/>
52      <param name="base_local_planner"
            value="base_local_planner/TrajectoryPlannerROS" />
53    </node>
54
55    <!-- TEB local planner -->
56    <node pkg="move_base" type="move_base" respawn="false"
          name="move_base_teb" output="screen" if="$(arg teb_planner)">
57      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_teb/costmap_common_params.yaml"
            command="load" ns="global_costmap" />
58      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_teb/costmap_common_params.yaml"
            command="load" ns="local_costmap" />
59      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_teb/local_costmap_params.yaml"
            command="load" />
60      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_teb/global_costmap_params.yaml"
            command="load" />
61      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_teb/local_planner_params.yaml"
            command="load" />
62
63      <param name="controller_frequency" value="10.0"/>
```

```
64     <param name="base_local_planner"
           value="teb_local_planner/TebLocalPlannerROS" />
65   </node>
66
67 </launch>
```

move_base.launch

```
1 <launch>
2
3   <node pkg="map_server" type="map_server" args="$(find
        tfg_roomba)/map/home_multi.yaml" respawn="true" name="map_server" >
4     <param name="frame_id" value="map" />
5   </node>
6
7    <!-- Show everything in Rviz -->
8   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
        tfg_roomba)/config/multi.rviz"
9     respawn="true"/>
10
11   <include file="$(find tfg_roomba)/launch/one_robot.launch" >
12     <arg name="my_ns" value="create_1" />
13     <arg name="x_init" value="0" />
14     <arg name="y_init" value="0" />
15   </include>
16
17   <include file="$(find tfg_roomba)/launch/one_robot.launch" >
18     <arg name="my_ns" value="create_2" />
19     <arg name="x_init" value="0" />
20     <arg name="y_init" value="0" />
21   </include>
22
23 </launch>
```

m3_autonomous_multi.launch

```
1 <launch>
2
3   <arg name="my_ns" default="roomba"/>
```

```
4   <arg name="x_init" default="0"/>
5   <arg name="y_init" default="0"/>
6
7   <group ns="$(arg my_ns)">
8     <include file="$(find tfg_roomba)/launch/move_base_multi.launch" >
9       <arg name="my_ns" value="$(arg my_ns)" />
10      <arg name="x_init" value="$(arg x_init)" />
11      <arg name="y_init" value="$(arg y_init)" />
12    </include>
13  </group>
14
15 </launch>
```

one_robot.launch

```
1 <launch>
2
3   <arg name="my_ns" default=""/>
4   <arg name="x_init" default="0"/>
5   <arg name="y_init" default="0"/>
6
7 <!-- Publish scans from best pose at a max of 10 Hz -->
8   <node pkg="amcl" type="amcl" name="amcl" output="screen"
        args="scan:=/$(arg my_ns)/scan map:=/map">
9     <param name="use_map_topic" value="true"/>
10    <param name="base_frame_id" value="$(arg my_ns)/base_link"/>
11    <param name="odom_model_type" value="diff"/>
12    <param name="odom_alpha5" value="0.1"/>
13    <param name="transform_tolerance" value="0.3" />
14    <param name="gui_publish_rate" value="10.0"/>
15    <param name="laser_max_beams" value="30"/>
16    <param name="min_particles" value="100"/>
17    <param name="max_particles" value="1000"/>
18    <param name="kld_err" value="0.05"/>
19    <param name="kld_z" value="0.99"/>
20    <param name="odom_alpha1" value="0.2"/>
21    <param name="odom_alpha2" value="0.2"/>
22    <param name="odom_alpha3" value="0.8"/>
23    <param name="odom_alpha4" value="0.2"/>
```

```
24      <param name="laser_z_hit" value="0.5"/>
25      <param name="laser_z_short" value="0.05"/>
26      <param name="laser_z_max" value="0.05"/>
27      <param name="laser_z_rand" value="0.5"/>
28      <param name="laser_sigma_hit" value="0.2"/>
29      <param name="laser_lambda_short" value="0.1"/>
30      <param name="laser_lambda_short" value="0.1"/>
31      <param name="laser_model_type" value="likelihood_field"/>
32      <param name="laser_likelihood_max_dist" value="2.0"/>
33      <param name="update_min_d" value="0.2"/>
34      <param name="update_min_a" value="0.5"/>
35      <param name="odom_frame_id" value="$(arg my_ns)/odom"/>
36      <param name="resample_interval" value="1"/>
37      <param name="recovery_alpha_slow" value="0.0"/>
38      <param name="recovery_alpha_fast" value="0.0"/>
39      <param name="initial_pose_x" value="$(arg x_init)" />
40      <param name="initial_pose_y" value="$(arg y_init)" />
41    </node>
42
43    <node pkg="move_base" type="move_base" respawn="false"
          name="move_base" output="screen">
44      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_teb_multi/costmap_common_params.yaml"
            command="load" ns="global_costmap" subst_value="true">
45        <arg name="my_ns" value="$(arg my_ns)" />
46      </rosparam>
47      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_teb_multi/costmap_common_params.yaml"
            command="load" ns="local_costmap" subst_value="true">
48        <arg name="my_ns" value="$(arg my_ns)" />
49      </rosparam>
50      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_teb_multi/local_costmap_params.yaml"
            command="load" subst_value="true">
51        <arg name="my_ns" value="$(arg my_ns)" />
52      </rosparam>
53      <rosparam file="$(find
            tfg_roomba)/config/move_base_config_teb_multi/global_costmap_params.yaml"
            command="load" subst_value="true">
```

```
54        <arg name="my_ns" value="$(arg my_ns)" />
55      </rosparam>
56      <rosparam file="$(find
          tfg_roomba)/config/move_base_config_teb_multi/local_planner_params.yaml"
          command="load" subst_value="true"/>
57
58      <param name="controller_frequency" value="10.0"/>
59      <param name="base_local_planner"
          value="teb_local_planner/TebLocalPlannerROS" />
60    </node>
61
62  </launch>
```

move_base_multi.launch

C.3. Paquete tfg_remote

Carpeta launch

```
1  <?xml version="1.0"?>
2  <launch>
3
4    <arg name="config" default="$(find tfg_remote)/config/multi.yaml" />
5    <arg name="desc" default="true" />
6
7    <node name="ca_driver" pkg="ca_driver" type="ca_driver"
         output="screen">
8      <rosparam command="load" file="$(arg config)" />
9      <param name="robot_model" value="CREATE_2" />
10     <param name="dev" value="/dev/ttyUSB1" />
11   </node>
12
13   <!-- Robot description -->
14   <param name="robot_description" command="$(find xacro)/xacro.py
         '$(find ca_description)/urdf/create_2.urdf.xacro'" />
15
16   <node name="robot_state_publisher" pkg="robot_state_publisher"
         type="robot_state_publisher" output="screen" >
17     <param name="tf_prefix" value="create_1"/>
```

```
18  </node>
19
20  <node name="rplidarNode" pkg="rplidar_ros" type="rplidarNode"
        output="screen" args="scan:=/create_1/scan">
21    <param name="serial_port" type="string" value="/dev/ttyUSB0"/>
22    <param name="serial_baudrate" type="int" value="256000"/><!--A3 -->
23    <param name="frame_id" type="string" value="create_1/lidar"/>
24    <param name="inverted" type="bool" value="false"/>
25    <param name="angle_compensate" type="bool" value="true"/>
26    <param name="scan_mode" type="string" value="Sensitivity"/>
27  </node>
28
29  <node pkg="tf2_ros" type="static_transform_publisher" name="lidar_tf"
        args="-0.01 0.105 0.055 3.041592 0 0 create_1/base_link
        create_1/lidar" />
30
31 </launch>
```

r1_slam.launch

```
1 ?xml version="1.0"?>
2 <launch>
3
4    <group ns="create_1">
5
6      <arg name="config" default="$(find tfg_remote)/config/multi.yaml"
          />
7      <arg name="desc" default="true" />
8
9      <node name="ca_driver" pkg="ca_driver" type="ca_driver"
          output="screen">
10  <rosparam command="load" file="$(arg config)" />
11  <param name="robot_model" value="CREATE_2" />
12  <param name="dev" value="/dev/ttyUSB0" />
13     </node>
14
15     <!-- Robot description -->
16     <param name="robot_description" command="$(find xacro)/xacro.py
          '$(find ca_description)/urdf/create_2.urdf.xacro'" />
```

```
17
18      <node name="robot_state_publisher" pkg="robot_state_publisher"
           type="robot_state_publisher" output="screen" >
19  <param name="tf_prefix" value="create_1"/>
20      </node>
21
22  <node name="rplidarNode" pkg="rplidar_ros" type="rplidarNode"
        output="screen" args="scan:=/create_1/scan">
23    <param name="serial_port" type="string" value="/dev/ttyUSB1"/>
24    <param name="serial_baudrate" type="int" value="256000"/><!--A3 -->
25    <param name="frame_id" type="string" value="create_1/lidar"/>
26    <param name="inverted" type="bool" value="false"/>
27    <param name="angle_compensate" type="bool" value="true"/>
28    <param name="scan_mode" type="string" value="Sensitivity"/>
29  </node>
30
31  <node pkg="tf2_ros" type="static_transform_publisher" name="lidar_tf"
        args="-0.01 0.105 0.055 3.041592 0 0 create_1/base_link
        create_1/lidar" />
32
33  </group>
34
35 </launch>
```

r2_autonomous.launch

Bibliografía

Ackerman, E. (2020). Autonomous Robots Are Helping Kill Coronavirus in Hospitals. *IEEE Spectrum*. Recuperado el 19 de mayo de 2020, desde https://spectrum.ieee.org/automaton/robotics/medical-robots/autonomous-robots-are-helping-kill-coronavirus-in-hospitals

Ahmadzadeh, S. y Ghanavati, M. (2012). Navigation of mobile robot using the PSO particle swarm optimization. *Journal of Academic and Applied Studies (JAAS)*, *2*(1), 32-38.

Balch, T. (1993). Avoiding the past: A simple but effective strategy for reactive navigation, En *Proceedings IEEE International Conference on Robotics and Automation*. IEEE.

Berglund, T., Brodnik, A., Jonsson, H., Staffanson, M. y Soderkvist, I. (2009). Planning smooth and obstacle-avoiding B-spline paths for autonomous mining vehicles. *IEEE Transactions on Automation Science and Engineering*, *7*(1), 167-172.

Blanco-Claraco, J. L. (2019). A Modular Optimization Framework for Localization and Mapping. *Proc. of Robotics: Science and Systems (RSS)*, *2*.

Blanco-Claraco, J. L., Fernández-Madrigal, J. A. y Gonzalez, J. (2008). Towards a Unified Bayesian Approach to Hybrid Metric-Topological SLAM. *IEEE Transactions on Robotics*, *24*(2), 259-270.

Borenstein, J. y Koren, Y. (1989). Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on systems, Man, and Cybernetics*, *19*(5), 1179-1187.

Bradski, G. y Kaehler, A. (2008). *Learning OpenCV*. O'Reilly Media, Inc.

Brian Gerkey. (2020a). *AMCL package*. Recuperado el 7 de abril de 2020, desde http://wiki.ros.org/amcl

Brian Gerkey. (2020b). *Gmapping package*. Recuperado el 7 de abril de 2020, desde http://wiki.ros.org/gmapping

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE J. Robot Autom.*, *2*, 14-23.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I. y Leonard, J. (2016). Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age. *IEEE Transactions on Robotics*, *32*(6), 1309-1332.

Cai, C. y Ferrari, S. (2009). Information-driven sensor path planning by approximate cell decomposition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *39*(3), 672-689.

Canonical Ltd. (2020a). *Ubuntu 18.04 Server*. Recuperado el 5 de abril de 2020, desde https://ubuntu.com/download/raspberry-pi/thank-you?version=18.04.4&architecture=arm64+raspi3

Canonical Ltd. (2020b). *Ubuntu 18.04.4 LTS (Bionic Beaver)*. Recuperado el 5 de abril de 2020, desde http://releases.ubuntu.com/bionic/

Canonical Ltd. (2020c). *Ubuntu Mate*. Recuperado el 5 de abril de 2020, desde https://ubuntu-mate.org/download/armhf/bionic/

Carlone, L., Calafiore, G., Tommolillo, C. y Dellaert, F. (2016). Planar Pose Graph Optimization: Duality, Optimal Solutions, and Verification. *IEEE Transactions on Robotics*, *32*(3), 545-565.

Chen, C., Li, H.-X. y Dong, D. (2008). Hybrid control for robot navigation-a hierarchical Q-learning algorithm. *IEEE Robotics & Automation Magazine*, *15*(2), 37-47.

Choi, J.-w., Curry, R. y Elkaim, G. (2008). Path planning based on bézier curve for autonomous ground vehicles. IEEE.

Clark, C. M., Rock, S. M. y Latombe, J.-C. (2003). Motion planning for multiple mobile robots using dynamic networks, En *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*. IEEE.

Connell, J. H. Et al. (1992). SSS: A hybrid architecture applied to robot navigation., En *Proceedings of the IEEE Conference on Robotics and Automation (ICRA)*.

Costante, G., Mancini, M., Valigi, P. y Ciarfuglia, T. A. (2016). Exploring Representation Learning With CNNs for Frame-to-Frame Ego-Motion Estimation. *IEEE Robotics and Automation Letters*, *1*(1), 18-25.

Dias, M. B. y Stentz, A. (2002). Opportunistic optimization for market-based multirobot control, En *IEEE/RSJ international conference on intelligent robots and systems*. IEEE.

Dijkstra, E. W. Et al. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269-271.

Dissanayake, G., Huang, S., Wang, Z. y Ranasinghe, R. (2011). A review of recent developments in Simultaneous Localization and Mapping, En *International Conference on Industrial and Information Systems*, IEEE.

Durrant-Whyte, H. y Bailey, T. (2006a). Simultaneous Localisation and Mapping (SLAM): Part I. The essential algorithms. *IEEE Robotics and Automation Magazine*, *13*(2), 99-110.

Durrant-Whyte, H. y Bailey, T. (2006b). Simultaneous Localisation and Mapping (SLAM): Part II. The state of the art. *Robotics and Autonomous Systems (RAS)*, *13*(3), 108-117.

Fairchild, C. y Harman, T. L. (2017). *ROS Robotics by example* (2.a ed.). Packt Publishing.

Fukuda, T. y Kawauchi, Y. (1990). Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator, En *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE.

Galisteo, A. (2016). Así funcionan los robots que trabajan para Amazon. *Expansión*. Recuperado el 19 de mayo de 2020, desde https://www.expansion.com/pymes/2016/10/16/58010ec2e5fdea14668b45b5.html

Gerkey, B. P. y Matarić, M. J. (2000). Murdoch: Publish/subscribe task allocation for heterogeneous agents.

González-Nalda, P., Calvo, I., Etxeberria-Agiriano, I., Zulueta, E. y López-Guede, J. M. (2015). Hacia un framework basado en ROS para la implementación de Sistemas Ciberfísicos, En *Actas de las XXXVI Jornadas de Automática*, Comité Español de Automática de la IFAC (CEA-IFAC). Bilbao, País Vasco, España.

Google LLC. (s.f.). *Tensorflow library*. Recuperado el 21 de mayo de 2020, desde https://www.tensorflow.org/

Grupo de investigación ”Automática, Robótica y Mecatrónica” de la UAL. (2020a). *Planificación multi-robot iCreate*. Recuperado el 29 de mayo de 2020, desde https://www.youtube.com/watch?v=OlcR57Hr1OI

Grupo de investigación ”Automática, Robótica y Mecatrónica” de la UAL. (2020b). *Planificación multi-robot iCreate con obstáculos*. Recuperado el 29 de mayo de 2020, desde https://www.youtube.com/watch?v=MOgLps_AGko

Grupo de investigación ”Automática, Robótica y Mecatrónica” de la UAL. (2020c). *Simulación planificación multi-robot en invernadero*. Recuperado el 2 de junio de 2020, desde https://www.youtube.com/watch?v=Q8LzVKN32a4

Gudise, V. G. y Venayagamoorthy, G. K. (2003). Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks, En *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS’03 (Cat. No. 03EX706)*. IEEE.

Gupta, J. K., Egorov, M. y Kochenderfer, M. (2017). Cooperative multi-agent control using deep reinforcement learning, En *International Conference on Autonomous Agents and Multiagent Systems*. Springer.

Gutmann, J. S. y Konolige, K. (1999). Incremental Mapping of Large Cyclic Environments, En *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, IEEE.

Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network, En *Neural networks for perception*. Elsevier.

Huh, U.-Y. y Chang, S.-R. (2014). AG 2 continuous path-smoothing algorithm using modified quadratic polynomial interpolation. *International Journal of Advanced Robotic Systems*, *11*(2), 25.

HWAYEH. (2020). *Convertidor de DC-DC ajustable 5A 75W XL4015 módulo reductor 4,0-38 V a 1,25 V-36 V DIY fuente de alimentación ajustable*. Recuperado el 4 de abril de 2020, desde https://es.aliexpress.com/item/32697438446.html?spm=a2g0o.detail.1000060.1.32b2139euThO7a&gps-id=pcDetailBottomMoreThisSeller&scm=1007.13339.146401.0&scm_id=1007.13339.146401.0&scm-url=1007.13339.146401.0&pvid=6ad9e22f-170e-4b4e-814f-6de6912be5ea

International Federation of Robotics. (2019a). *Executive Summary World Robotics 2019 Service Robots*. Recuperado el 25 de abril de 2020, desde https://ifr.org/downloads/press2018/Executive_Summary_WR_Service_Robots_2019.pdf

International Federation of Robotics. (2019b). *IFR Press Conference*. Recuperado el 25 de abril de 2020, desde https://ifr.org/downloads/%20press2018/IFR%5C%20World%5C%20Robotics%5C%20Presentation%5C%20-%5C%2018%5C%20Sept%5C%202019.pdf

iRobot. (2018, 19 de julio). *iRobot ® Create ® 2 Open Interface (OI) Specification based on the iRobot ® Roomba ® 600*. iRobot Corporation.

Jan, G. E., Chang, K. Y. y Parberry, I. (2008). Optimal path planning for mobile robot navigation. *IEEE/ASME transactions on mechatronics*, *13*(4), 451-460.

Jones, E. G., Dias, M. B. y Stentz, A. (2011). Time-extended multi-robot coordination for domains with intra-path constraints. *Autonomous robots*, *30*(1), 41-56.

Joseph, R. G. L. (2019). *ROS Robotics Projects* (2.ª ed.). Packt Publishing Ltd.

Kala, R. (2016). Homotopic roadmap generation for robot motion planning. *Journal of Intelligent & Robotic Systems*, *82*(3-4), 555-575.

Kala, R., Shukla, A., Tiwari, R., Rungta, S. y Janghel, R. R. (2009). Mobile robot navigation control in moving obstacle environment using genetic algorithm, artificial neural networks and A* algorithm, En *2009 WRI World Congress on computer science and information engineering*. IEEE.

Karlik, B. y Olgac, A. V. (2011). Performance analysis of various activation functions in generalized MLP architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, *1*(4), 111-122.

Kendall, A., Grimes, M. y Cipolla, R. (2015). PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization, En *IEEE International Conference on Computer Vision (ICCV)*, IEEE.

Khamis, A. M., Elmogy, A. M. y Karray, F. O. (2011). Complex task allocation in mobile surveillance systems. *Journal of Intelligent & Robotic Systems*, *64*(1), 33-55.

Khamis, A., Hussein, A. y Elmogy, A. (2015). Multi-robot task allocation: A review of the state-of-the-art, En *Cooperative Robots and Sensor Networks 2015*. Springer.

Khatib, O. (1985). Real-time obstacle avoidance for manipulators and mobile robots, En *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. IEEE.

Kim, H., Leutenegger, S. y Davison, A. J. (2016). Real-Time 3D Reconstruction and 6-DoF Tracking with an Event Camera. *European Conference on Computer Vision (ECCV)*, 349-364.

Kiriy, E. y Buehler, M. (2002). Three-state extended kalman filter for mobile robot localization. *McGill University., Montreal, Canada, Tech. Rep. TR-CIM*, *5*, 23.

Koay, K. y Bugmann, G. (2004). Compensating intermittent delayed visual feedback in robot navigation, En *Proceedings of the IEEE Conference on Decision and Control*.

Kschischang, F. R., Frey, B. J. y Loeliger, H. A. (2001). Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, *47*(2), 498-519.

Küçük, S. (2012). *Serial and Parallel Robot Manipulators: Kinematics, Dynamics, Control and Optimization*. BoD–Books on Demand.

Kuderer, M., Sprunk, C., Kretzschmar, H. y Burgard, W. (2014). Online generation of homotopically distinct navigation paths, En *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

Li, Q.-L., Song, Y. y Hou, Z.-G. (2015). Neural network based FastSLAM for autonomous robots in unknown environments. *Neurocomputing*, *165*, 99-110.

Likhachev, M. y Ferguson, D. (2009). Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, *28*(8), 933-945.

Lindemann, S. R. y LaValle, S. M. (2005a). Smoothly blending vector fields for global robot navigation, En *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE.

Lindemann, S. R. y LaValle, S. M. (2005b). Smoothly blending vector fields for global robot navigation, En *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE.

Liu, C. y Kroll, A. (2012). A centralized multi-robot task allocation for industrial plant inspection by using a* and genetic algorithms, En *International Conference on Artificial Intelligence and Soft Computing*. Springer.

Long, P., Fanl, T., Liao, X., Liu, W., Zhang, H. y Pan, J. (2018). Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning, En *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.

Lozano-Pérez, T. y Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, *22*(10), 560-570.

Lu, D. (2020). *global_planner package*. Recuperado el 15 de mayo de 2020, desde http://wiki.ros.org/global_planner

Lu, F. y Milios, E. (1997). Globally consistent range scan alignment for environment mapping. *Autonomous Robots (AR)*, *4*(4), 333-349.

Makarenko, A., Brooks, A. y Kaupp, T. (2007). Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

Malagon-Soldara, S. M., Toledano-Ayala, M., Soto-Zarazua, G., Carrillo-Serrano, R. V. y Rivas-Araiza, E. A. (2015). Mobile Robot Localization: A Review of Probabilistic Map-Based Techniques. *International Journal of Robotics and Automation (IJRA)*, *4*(1), 73-81.

Manikas, T. W., Ashenayi, K. y Wainwright, R. L. (2007). Genetic algorithms for autonomous robot navigation. *IEEE Instrumentation & Measurement Magazine*, *10*(6), 26-31.

Marder-Eppstein, E. (2020). *nav_core package*. Recuperado el 15 de mayo de 2020, desde http://wiki.ros.org/nav_core

Marder-Eppstein, E. y Lu, D. (2020). *costmap_2d package*. Recuperado el 16 de mayo de 2020, desde http://wiki.ros.org/costmap_2d

Marder-Eppstein, E. y Perko, E. (2020). *base_local_planner package*. Recuperado el 16 de mayo de 2020, desde http://wiki.ros.org/base_local_planner

Massari, M., Giardini, G. y Bernelli-Zazzera, F. (2004). Autonomous navigation system for planetary exploration rover based on artificial potential fields, En *Proceedings of Dynamics and Control of Systems and Structures in Space (DCSSS) 6th Conference*.

Mechatronics Art. (2020). *iRobot iCreate 2 Reference CAD Model*. Recuperado el 21 de mayo de 2020, desde https://grabcad.com/library/irobot-icreate-2-reference-cad-model-1

Melo, L. F. D. y Junior, J. F. M. (2010). Trajectory planning for nonholonomic mobile robot using extended Kalman filter. *Mathematical problems in engineering*, *2010*.

Metropolis, N. y Ulam, S. (1949). The monte carlo method. *Journal of the American statistical association*, *44*(247), 335-341.

Mohanty, P. K. y Parhi, D. R. (2013). Controlling the motion of an autonomous mobile robot using various techniques: a review. *Journal of Advance Mechanical Engineering*, *1*(1), 24-39.

Montemayor, G. y Wen, J. T. (2005). Decentralized collaborative load transport by multiple robots, En *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE.

Montemerlo, M., Roy, N. y Thrun, S. (2003). Perspectives on standardization in mobile robot programming: The Carnegie Mellon Navigation (CARMEN) Toolkit, En *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, Nevada.

Moreno, J. C., Giménez, A. y Rodríguez, F. (2019). Proyecto AGRICOBIOT: Robot Colaborativo para Transporte Inteligente en Interior de Invernaderos con Soporte en IoT, En *Jornadas Nacionales de Robótica 2019 (JNR 2019)*. Comité Español de Automática (CEA). Alicante.

Moulard, T. (2011). *Coordinate Frames for Humanoid Robots (REP 120)*, En *ROS Enhancement Proposals (REP)*.

Nakhaeinia, D., Tang, S. H., Noor, S. M. y Motlagh, O. (2011). A review of control architectures for autonomous navigation of mobile robots. *International Journal of the Physical Sciences*, *6*(2), 169-174.

Negenborn, R. (2003). Robot localization and kalman filters. *Utrecht Univ., Utrecht, Netherlands, Master's thesis INF/SCR-0309*.

Open Robotics. (2020a). *create_autonomy package summary*. Recuperado el 6 de abril de 2020, desde http://wiki.ros.org/create_autonomy

Open Robotics. (2020b). *Gazebo plugins*. Recuperado el 8 de abril de 2020, desde http://gazebosim.org/tutorials?tut=ros_gzplugins

Open Robotics. (2020c). *Gazebo: Robot simulation made easy*. Recuperado el 6 de abril de 2020, desde http://gazebosim.org/

Open Robotics. (2020d). *ROS Distributions*. Recuperado el 6 de abril de 2020, desde http://wiki.ros.org/Distributions

Open Robotics. (2020e). *ROS Parameter Server*. Recuperado el 6 de abril de 2020, desde %7Bhttp://wiki.ros.org/Parameter%20Server%7D

Open Robotics. (2020f). *ROS Services*. Recuperado el 6 de abril de 2020, desde http://wiki.ros.org/Services

Open Robotics. (2020g). *Xacro package summary*. Recuperado el 7 de abril de 2020, desde http://wiki.ros.org/xacro

Parker, L. E. (1994). ALLIANCE: An architecture for fault tolerant, cooperative control of heterogeneous mobile robots, En *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*. IEEE.

Patle, B., Pandey, A., Parhi, D., Jagadeesh, A. Et al. (2019). A review: On path planning strategies for navigation of mobile robot. *Defence Technology*.

Paul, A. S. y Wan, E. A. (2008). Wi-Fi based indoor localization and tracking using sigma-point Kalman filtering methods, En *2008 IEEE/ION Position, Location and Navigation Symposium*. IEEE.

Perron, J. (2020a). *create_autonomy package*. Recuperado el 16 de mayo de 2020, desde http://wiki.ros.org/create_autonomy

Perron, J. (2020b). *libcreate library*. Recuperado el 16 de mayo de 2020, desde http://wiki.ros.org/libcreate

Pfaff, P., Burgard, W. y Fox, D. (2006). Robust monte-carlo localization using adaptive likelihood models, En *European robotics symposium 2006*. Springer.

Pol, R. S. y Murugan, M. (2015). A review on indoor human aware autonomous mobile robot navigation through a dynamic environment survey of different path planning algorithm and methods, En *2015 International Conference on Industrial Instrumentation and Control (ICIC)*. IEEE.

Prorok, A., Bahr, A. y Martinoli, A. (2012). Low-cost collaborative localization for large-scale multi-robot systems, En *2012 IEEE International Conference on Robotics and Automation*. Ieee.

Pygame. (2020). *Pygame Documentation*. Recuperado el 8 de abril de 2020, desde https://www.pygame.org/docs/

Quigley, M., Berger, E. y Ng, A. Y. (2007). STAIR: Hardware and Software Architecture. *AAAI 2007 Robotics Workshop*.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R. y Ng, A. (2009). ROS: an open-source Robot Operating System, 1-3.

Quigley, M., Gerkey, B. y Smart, W. D. (2015). *Programming Robots with ROS* (2.ª ed.). O'Reilly Media, Inc.

Raja, P. y Pugazhenthi, S. (2012). Optimal path planning of mobile robots: A review. *International journal of physical sciences*, *7*(9), 1314-1320.

Raspberry Pi Foundation. (2020a). *NOOBS Documentation*. Recuperado el 5 de abril de 2020, desde https://www.raspberrypi.org/documentation/installation/noobs.md

Raspberry Pi Foundation. (2020b). *Raspberry Pi 3 Model B+*. Recuperado el 4 de abril de 2020, desde https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

Raspberry Pi Foundation. (2020c). *Raspbian Buster*. Recuperado el 5 de abril de 2020, desde https://www.raspberrypi.org/downloads/raspbian/

Ravankar, A., Ravankar, A. A., Kobayashi, Y., Hoshino, Y. y Peng, C.-C. (2018). Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges. *Sensors*, *18*(9), 3170.

Rosen, D. M., DuHadway, C. y Leonard, J. J. (2015). A convex relaxation for approximate global optimization in Simultaneous Localization and Mapping, En *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE.

Rösmann, C. (2020). *teb_local_planner package*. Recuperado el 16 de mayo de 2020, desde http://wiki.ros.org/teb_local_planner

Rösmann, C., Hoffmann, F. y Bertram, T. (2015). Planning of multiple robot trajectories in distinctive topologies, En *2015 European Conference on Mobile Robots (ECMR)*. IEEE.

Roumeliotis, S. I. y Bekey, G. A. (2000). Bayesian estimation and Kalman filtering: A unified framework for mobile robot localization, En *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. IEEE.

Schneider, E., Balas, O., Ozgelen, A. T., Sklar, E. I. y Parsons, S. (2014). An empirical evaluation of auction-based task allocation in multi-robot teams, En *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*.

Schneider, E., Sklar, E. I., Parsons, S. y Özgelen, A. T. (2015). Auction-based task allocation for multi-robot teams in dynamic environments, En *Conference Towards Autonomous Robotic Systems*. Springer.

Seraji, H. y Howard, A. (2002). Behavior-based robot navigation on challenging terrain: A fuzzy logic approach. *IEEE Transactions on Robotics and Automation*, *18*(3), 308-321.

Shi, Z., Wei, J., Wei, X., Tan, K. y Wang, Z. (2010). The task allocation model based on reputation for the heterogeneous multi-robot collaboration system, En *2010 8th World Congress on Intelligent Control and Automation*. IEEE.

Shiltagh, N. A. y Jalal, L. D. (2013). Optimal path planning for intelligent mobile robot navigation using modified particle swarm optimization. *International Journal of Engineering and Advanced Technology*, *2*(4), 260-267.

Slamtec. (2019, 25 de noviembre). *RPLIDAR A3. Low Cost 360 Degree Laser Range Scanner. Introduction and Datasheet*. Ver. 1.7. Shanghai Slamtec.Co.,Ltd.

Slamtec. (2020). *RPLIDAR package summary*. Recuperado el 7 de abril de 2020, desde http://wiki.ros.org/rplidar

Sleumer, N. y Tschichold-Gürmann, N. (1999). Exact cell decomposition of arrangements used for path planning in robotics. *Technical report/ETH Zürich, Department of Computer Science*, *329*.

Song, K.-T. y Sheen, L.-H. (2000). Heuristic fuzzy-neuro network and its application to reactive navigation of a mobile robot. *Fuzzy Sets and systems*, *110*(3), 331-340.

Sony Corporation. (2020). *Dualshock®3*. Recuperado el 8 de abril de 2020, desde https://www.playstation.com/es-es/explore/accessories/dualshock-3-wireless-controller/

Srinivasa, S., Ferguson, D. I., Weghe, M. V., Diankov, R., Berenson, D., Helfrich, C. y Strasdat, H. (2008). The robotic busboy: Steps towards developing a mobile robotic home assistant.

Szekeres, G. y Wilf, H. S. (1968). An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, *4*(1), 1-3.

Tan, Q., Fan, T., Pan, J. y Manocha, D. (2019). DeepMNavigate: Deep Reinforced Multi-Robot Navigation Unifying Local & Global Collision Avoidance. *arXiv preprint arXiv:1910.09441*.

Tanenbaum, A. S. y Bos, H. (2015). *Modern Operating Systems* (4.ª ed.). Pearson Prentice-Hall.

Tang, F. y Parker, L. E. (2005). Asymtre: Automated synthesis of multi-robot task solutions through software reconfiguration, En *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE.

Thomas, Z. (2020). Coronavirus: Will Covid-19 speed up the use of robots to replace human workers? *BBC*. Recuperado el 19 de mayo de 2020, desde https://www.bbc.com/news/technology-52340651

Thrun, S., Fox, D., Burgard, W. y Dellaert, F. (2001). Robust Monte Carlo localization for mobile robots. *Artificial intelligence*, *128*(1-2), 99-141.

U-Tech. (2020). *Batería LiPo U-TECH PRO 3s 11.1V 2200mAh 30C*. Recuperado el 4 de abril de 2020, desde https://rc-innovations.es/bateria-lipo-u-tech-pro-3s-11.1v-2200mah-30c-lipo-larga-duracion

Valentin, J., Niener, M., Shotton, J., Fitzgibbon, A., Izadi, S. y Torr, P. (2015). Exploiting Uncertainty in Regression Forests for Accurate Camera Relocalization, En *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE.

Wang, J., Gu, Y. y Li, X. (2012). Multi-robot task allocation based on ant colony algorithm. *Journal of Computers*, *7*(9), 2160-2167.

Wyobek, K., Berger, E., der Loos, H. V. y Salisbury, K. (2008). Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot, En *IEEE International Conference on Robotics and Automation (ICRA)*.

Yamashita, A., Arai, T., Ota, J. y Asama, H. (2003). Motion planning of multiple mobile robots for cooperative manipulation and transportation. *IEEE Transactions on Robotics and Automation*, *19*(2), 223-237.

Yang, G.-Z., Nelson, B. J., Murphy, R. R., Choset, H., Christensen, H., Collins, S. H., Dario, P., Goldberg, K., Ikuta, K., Jacobstein, N. Et al. (2020). Combating COVID-19—The role of robotics in managing public health and infectious diseases. Science Robotics.

Zadeh, L. A. (1965). Fuzzy sets. *Information and control*, *8*(3), 338-353.

Zadeh, L. A. (1988). Fuzzy logic. *Computer*, *21*(4), 83-93.

Zhu, Z., Schmerling, E. y Pavone, M. (2015). A convex optimization approach to smooth trajectories for motion planning with car-like robots, En *2015 54th IEEE Conference on Decision and Control (CDC)*. IEEE.

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL, 2019/2020