UNIVERSITAT POLITÈCNICA DE VALÈNCIA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA
INSTITUTO DE AUTOMÁTICA E INFORMÁTICA INDUSTRIAL

INTEGRACIÓN DE SISTEMAS MULTI-AGENTE EN PLATAFORMAS EMBEBIDAS HETEROGÉNEAS CON RECURSOS LIMITADOS PARA TAREAS DE LOCALIZACIÓN Y COORDINACIÓN EN DETECCIÓN Y EVASIÓN DE COLISIONES EN ROBÓTICA MÓVIL.

TESIS DOCTORAL REALIZADA POR:

Ángel Soriano Vigueras

DIRIGIDA POR:

Dr. Ángel Valera Fernández
Dra. Marina Vallés Miquel

Valencia, Julio 2017

A mi familia y amigos.

A Rosa.

Un agente omnisciente conoce el resultado de su acción y actúa de acuerdo con él; sin embargo, en realidad la omnisciencia no es posible.

Stuart J. Russell.

RESUMEN

En los últimos años, dentro del campo de la robótica móvil, las arquitecturas distribuidas se están convirtiendo en el pilar fundamental sobre el cual desarrollar algoritmos modulares, escalables y reaprovechables que ofrezcan soluciones flexibles aplicables fácil y rápidamente a cualquier plataforma con unos cambios mínimos en el software, evitando de este modo reinventar la rueda enfrentándose al mismo problema una y otra vez con cada tipo distinto de robot sobre el que se desea trabajar.

Dentro de este tipo de arquitectura los sistemas multi-agente permiten orientar su programación hacia comportamientos modulares y ofrecen una gestión de comunicaciones bien definida. La habilidad de que un grupo de robots pueda comunicarse entre sí con cierta facilidad, otorga información nueva a los agentes que el hardware de sensorización puede en ocasiones no ser capaz de detectar, y que puede ser utilizada para proponer nuevas soluciones alternativas o complementarias a las ya definidas dentro del área. Este es exactamente el motivo y el reto al que se enfrenta este trabajo, aplicado a robots móviles de recursos limitados.

La primera aportación de esta tesis radica en el desarrollo de un middleware de control el cual permite el uso de dos importantes frameworks distribuidos: *JADE* orientado al desarrollo software multi-agente y *ROS* orientado a software robótico distribuido. El middleware se integró en el conjunto de robots disponibles para pruebas y se utiliza como base de la arquitectura de la mayoría de experimentos simulados y prácticos que se presentan.

Establecido y detallado el hardware y el software a utilizar, se establecen los nuevos algoritmos desarrollados para la localización local y global de robots móviles de recursos limitados, basados en filtros de fusión sensorial y en correspondencias geométricas. Además se evalúan las prestaciones utilizando los principios de control y muestreo basados en eventos y tomando como punto de partida los algoritmos en cascada basados en el tiempo.

Partiendo de dichos algoritmos de localización, se describen metodologías de navegación y coordinación de grupos de robots cooperativos, donde las comunicaciones

entre agentes son la base del éxito y donde se observan resultados satisfactorios tanto a nivel individual por robot, como a nivel global sobre el conjunto de robots.

Cuando se trabaja sobre un escenario compartido por diversos robots móviles navegando, una de las problemáticas más críticas es conseguir que los robots no choquen entre ellos. Por este motivo, otra de las aportaciones más importantes de este trabajo ha sido el desarrollo de un algoritmo de detección y evasión de colisiones basado también en el consenso y el acuerdo entre robots a través de las comunicaciones punto a punto entre ellos, el cual se ha definido tanto para situaciones donde un robot colisionaba sólo con otro robot, como para casos en los que ocurrían múltiples colisiones al mismo tiempo.

Finalmente se exponen las conclusiones y las posibles líneas de trabajo sobre las que seguir investigando y desarrollando a partir de los resultados obtenidos en la presente tesis.

Palabras clave: Robótica móvil, arquitectura distribuida, sistemas multi-agente, filtros de Kalman distribuidos, odometría, navegación cooperativa, localización geométrica, localización cooperativa de robots, coordinación de robots, comunicación basada en eventos, evasión de colisiones, *LEGO Mindstorms NXT*, *Summit XL*, *e-puck*.

Subvenciones: Este trabajo ha sido financiado parcialmente por el Ministerio de Ciencia e Innovación de España bajo los proyectos FEDER/CICYT de investigación DPI2008-06737-C02-01 y DPI2010-20814-C02-02.

RESUM

En els últims anys, dins del camp de la robòtica mòbil, les arquitectures distribuïdes s'estan convertint en el pilar fonamental sobre el qual desenvolupar algorismes modulars, escalables i reaprovechables que oferisquen solucions flexibles aplicables fàcil i ràpidament a qualsevol plataforma amb uns canvis mínims en el programari, evitant d'aquesta manera reinventar la roda enfrontant-se al mateix problema una vegada i una altra amb cada tipus diferent de robot sobre el qual es desitja treballar.

Dins d'aquest tipus d'arquitectura els sistemes multi-agent permeten orientar la seua programació cap a comportaments modulars i ofereixen una gestió de comunicacions ben definida. L'habilitat de que un grup de robots puga comunicar-se entre si amb certa facilitat, atorga informació nova als agents que el hardware de sensorització pot en ocasions no ser capaç de detectar, i que pot ser utilitzada per a proposar noves solucions alternatives o complementàries a les ja definides dins de l'àrea. Aquest és exactament el motiu i el repte al que s'enfronta aquest treball, aplicat a robots mòbils de recursos limitats.

La primera aportació d'aquesta tesi radica en el desenvolupament d'un middleware de control el qual permet l'ús de dos importants frameworks distribuïts: *JADE* orientat al desenvolupament de la programació multi-agent i *ROS* orientat a la programació robòtica distribuïda. El middleware es va integrar en el conjunt de robots disponibles per a proves i s'utilitza com a base de l'arquitectura de la majoria d'experiments simulats i pràctics que es presenten.

Establit i detallat el hardware i el software a utilitzar, s'estableixen els nous algorismes desenvolupats per a la localització local i global de robots mòbils de recursos limitats, basats en filtres de fusió sensorial i en correspondències geomètriques. A més s'avaluen les prestacions utilitzant els principis de control i mostreig basats en esdeveniments i prenent com a punt de partida els algorismes en cascada basats en el temps.

Partint d'aquests algorismes de localització, es descriuen metodologies de navegació i coordinació de grups de robots cooperatius, on les comunicacions entre agents

són la base de l'èxit i on s'observen resultats satisfactoris tant a nivell individual per robot, com a nivell global sobre el conjunt de robots.

A més, quan es treballa sobre un escenari compartit per diversos robots mòbils navegant, una de les problemàtiques més crítiques és aconseguir que els robots no xoquen entre ells. Per aquest motiu, una altra de les aportacions més importants d'aquest treball ha sigut el desenvolupament d'un algorisme de detecció i evasió de col·lisions basat també en el consens i l'acord entre robots a través de les comunicacions punt a punt entre ells, el qual s'ha definit tant per a situacions on un robot col·lisionava només amb un altre robot, com per a casos en els quals ocorrien múltiples col·lisions al mateix temps.

Finalment s'exposen les conclusions i les possibles línies de treball sobre les quals seguir investigant i desenvolupant a partir dels resultats obtinguts en la present tesi.

Paraules clau: Robòtica mòbil, arquitectura distribuïda, sistemes multi-agent, filtres de Kalman distribuïts, odometría, navegació cooperativa, localització geomètrica, localització cooperativa de robots, coordinació de robots, comunicació basada en esdeveniments, evasió de col·lisions, *LEGO Mindstorms NXT*, *Summit XL*, *e-puck*.

Subvencions: Aquest treball ha sigut finançat parcialment pel Ministeri de Ciència i Innovació d'Espanya sota els projectes de recerca FEDER/CICYT DPI2008-06737-C02-01 i DPI2010-20814-C02-02.

ABSTRACT

In recent years, within the field of mobile robotics, distributed architectures are becoming the fundamental pillar on which to develop modular, scalable and reusable algorithms that offer flexible solutions that can be applied easily and quickly to any platform with minimal changes in the software. Thus avoiding to reinvent the wheel facing the same problem again and again with each different type of robot on which it wants to work.

Within this type of architecture, the multi-agent systems have the ability to orient their programming to modular behaviors and offer a well defined communication management. The ability for a group of robots to communicate with each other with some ease, gives new information to the agents that the sensoring hardware may sometimes not be able to detect, and that can be used to propose new or complementary solutions to the already defined within the area. This is exactly the reason and the challenge faced by this work, applied to mobile robots with limited resources.

The first contribution of this thesis lies in the development of a control middleware which allows the use of two important distributed frameworks: *JADE* oriented multi-agent software development and *ROS* oriented distributed robotic software. The middleware was integrated into the set of robots available for testing and it was used as the basis of the architecture of most simulated and practical experiments that are presented.

Once the established and detailed the hardware and software used, the new algorithms developed for local and global localization of mobile robots with limited resources, based on filters of sensory fusion and geometric correspondences are established. In addition, the benefits are evaluated using the principles of control and sampling based on events and taking as a starting point the cascade algorithms based on time.

Based on these localization algorithms, navigation and coordination methodologies are described for groups of cooperative robots, where the communications between

agents are the basis of success and where satisfactory results are observed both individually by robot and globally on the set Of robots.

When working on a scenario shared by various mobile robots navigating, one of the most critical issues is to get the robots do not collide with each other. For this reason, another of the most important contributions of this work has been the development of an algorithm of detection and avoidance of collisions based on the consensus and agreement between robots through the point-to-point communications between them. For cases when a robot collided with just another robot, and for cases where multiple collisions occurred at the same time.

Finally, the conclusions and the possible lines of work on which to continue researching and developing from the results obtained in this thesis are presented.

Keywords: Mobile robotics, distributed architecture, multi-agent systems, distributed Kalman filters, odometry, cooperative navigation, geometric localization, robot co-location, robot coordination, event-based communication, collision avoidance, *LEGO Mindstorms NXT*, *Summit XL*, *e-puck*.

Acknowledgements: This work has been partially funded by FEDER / CICYT research projects with references DPI2008-06737-C02-01 and DPI2010-20814-C02-02 financed by Ministerio de Ciencia e Innovación (Spain).

Agradecimientos

Esta tesis ha sido una grata experiencia personal que no habría podido llevado a cabo sin la colaboración de las muchas personas que de una manera u otra me han ayudado, apoyado y animado a ser constante, paciente y superar los obstáculos que se han ido presentando a lo largo de su desarrollo.

En primer lugar, me gustaría expresar todo mi agradecimiento a mis directores de tesis Dr. Ángel Valera Fernández y Dra. Marina Vallés Miquel por su ayuda, su paciencia y el apoyo ofrecido en todo momento tanto en los temas académicos, como en los personales. Son muchas las aventuras que hemos recorrido en esta investigación y en parte a ellos les debo parte de mi inquietud, mi curiosidad, mi ilusión y mi ambición personal por el campo de la investigación en la robótica.

También me gustaría agradecer a todos los responsables de los proyectos de investigación donde he podido colaborar de manera activa durante la realización de esta tesis: Dr. Pedro Albertos Pérez, Dr. Martín Mellado Arteche y Dr. Julián José Salt Llobregat. Gracias a ellos he podido estar implicado en proyectos financiados por el Ministerio de Economía y Competitividad del gobierno de España, con referencias DPI 2008-06737-C02-01 (Núcleo de Control en Sistemas distribuidos.), DPI2011-28507-C02-01 (Desarrollo de Controladores Basado en Misiones.), TEC2012-31506 (Desarrollo y Validación de Técnicas de Ahorro Energético en Aplicaciones de Control.), y el proyecto Desarrollo de una Estación Prototipo Multi-robot de Manipulación del Producto de Dulcesol Tortas. Gracias a dichos proyectos, he podido financiar mis estudios y mis publicaciones.

Quiero agradecer también la dedicación a esta tesis por parte del resto de profesores del Instituto de Automática e Informática Industrial de la Universidad Politécnica de Valencia, en especial a Dr. Enrique Bernabeu Soler con quien he compartido varias publicaciones y a Dr. Antonio Sánchez Salmerón quien me ha dedicado su tiempo y paciencia.

Por otra parte, quiero agradecer a la Universidad Politécnica de Valencia y al Instituto de Automática e Informática Industrial, haberme permitido realizar mi investigación en sus instalaciones y poner a mi disposición los diversos equipos

requeridos para la realización de las numerosas pruebas experimentales, así como por brindarme la oportunidad de ejercer como profesor asociado en varias asignaturas del departamento, DISA.

Quiero además mostrar mi agradecimiento a los compañeros del AI2 que de una forma u otra me han alentado en el desarrollo de esta tesis, a Javier Gómez, a Eugenio Iborra, Marc Bosch, Carlos Blanes, Pablo Beltrán, Miguel Ángel Martínez, José Cazalilla, Borja Ponz, Carlos Sánchez, Leonardo Marín y el resto de compañeros junto con los que día a día hemos ido avanzando apoyándonos entre nosotros.

Por último pero no menos importante, me gustaría agradecer el apoyo incondicional que durante todo este tiempo me han brindado mis padres, mi hermana, mis tíos, mi familia y mis amigos, que siempre han estado pendientes de mí y preocupándose por saber qué tal me iba en este viaje. Gracias por ayudarme en los momentos en los que más lo necesité. En especial a Rosa, compañera colateral de esta gran aventura, por su paciencia, cariño y apoyo constante y permanente.

Muchas gracias a todos,

Ángel Soriano Vigueras

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1

Introducción

Con el objetivo de contextualizar la temática de la presente tesis sobre el uso de sistemas multi-agente en tareas de localización, evasión de colisiones y coordinación en robots móviles de recursos limitados, a continuación se describen los aspectos generales asociados a las arquitecturas robóticas, a la teoría básica de navegación de robots móviles heterogéneos en un escenario compartido, así como la justificación y los objetivos planteados en la realización de la presente tesis.

1.1 Aspectos generales

Hoy en día se diseñan y construyen robots móviles para multitud de aplicaciones. Muchos hogares disponen de un robot móvil como herramienta auxiliar de limpieza, cada día se observan los resultados de grandes empresas intentando ofrecer un vehículo móvil autónomo con garantías de seguridad vial, en hospitales se han instalado plataformas móviles para el transporte de camillas y medicamentos, en grandes almacenes se utilizan robots móviles para la gestión y transporte de materiales, ... Cada vez se destinan más robots móviles a llevar a cabo procesos repetitivos de alta precisión en entornos controlados. Sin embargo, la tendencia actual es hacia sistemas robóticos que deben ser capaces de resolver problemas o misiones en entornos que son más complejos y menos controlados. Por este motivo, se necesitan sistemas robóticos que ofrezcan mayor autonomía e inteligencia.

En los equipos de robots móviles autónomos, donde un conjunto de robots funciona como un grupo para alcanzar un objetivo común, generalmente cada robot requiere de una estrategia de localización y navegación por el entorno evitando los posibles obstáculos que puedan encontrarse por el camino. Estas tareas deben realizarse de forma autónoma implementando un sistema de control a distintos niveles que

tenga en cuenta la sensorización del robot para determinar su estado (posición, orientación y caracterización del entorno que lo rodea). Además, estos sistemas multirobot, requieren habilidades sociales de cooperación para poder coordinarse y moverse en entornos no controlados dinámicos y complejos, donde la capacidad de comprender e interpretar el mundo que los rodea supone un importante reto a resolver.

Como consecuencia de estos retos, la complejidad de la arquitectura de software aumenta y las necesidades de computación normalmente se disparan. Por este motivo, la escalabilidad, la reutilización, la eficacia y la tolerancia a fallos que deben ofrecer las arquitecturas de software que gobiernan estos equipos juegan un papel fundamental. La arquitectura de software del sistema debe estar diseñada de forma distribuida y modular teniendo en cuenta los requisitos de tiempo real en sus niveles más bajos para garantizar su control correctamente.

Las principales características que la arquitectura de un software distribuido debe atender, se enumeran a continuación, [92, 60, 31]:

- *Concurrencia y arquitectura distribuida.* Debe ser capaz de tomar ventaja de todas las unidades de procesamiento disponibles en una forma concurrente (procesadores, multi-procesadores y microcontroladores) con el fin de cubrir todas las necesidades computacionales que el sistema autónomo necesite.
- *Modularidad.* La arquitectura de software debe estar formada por varios componentes de alta cohesión pero de bajo acoplamiento. Es decir, los componentes deben interactuar entre sí frecuentemente, sin embargo, las dependencias entre los mismos deben mantenerse mínimamente con el fin de obtener una arquitectura mantenible, escalable y reutilizable, que sea adaptable a los cambios y mejoras que puedan surgir.
- *Robustez y tolerancia a fallos.* El mal funcionamiento de un componente no debe bloquear completamente todo el sistema si no que en caso de ocurrir, el resto del sistema debe ser capaz de seguir trabajando de la mejor forma posible con los recursos disponibles para alcanzar el logro de los objetivos. Con este fin, el resto de los componentes (todavía en funcionamiento) debe ser capaz de actuar por su propia iniciativa y tomar decisiones de forma autónoma para superar estas situaciones. Estas decisiones deben tomarse sobre la base de la cooperación con otros agentes del sistema y sobre la información específica que poseen.
- *Eficiencia.* La eficiencia es un requisito fundamental en programación pero aún lo es más en tareas aplicadas a la robótica, especialmente cuando las plataformas sobre las que se trabaja disponen de recursos limitados y de una capacidad de cómputo reducida como es el caso de esta tesis.

Aunque estas características sean las deseadas en cualquier arquitectura de software, resultan especialmente importantes en el campo de la robótica, donde la falta de normas estándar internacionales y la cercanía con el hardware, han originado que el software desarrollado en robótica sea propenso a ser una solución acorde al robot sobre el que se desarrolla, de un solo uso, no escalable y poco modular.

A día de hoy han surgido distintos frameworks distribuidos que cumplen en cierta medida las necesidades mencionadas anteriormente. JADE [17] es un framework orientado a la programación distribuida de agentes que incorpora el protocolo estándar de comunicaciones FIPA-ACL [104]. Desde su aparición, ha mantenido un constante trabajo de mantenimiento y mejora lo cual ha promovido su uso en una gran variedad de campos muy dispares. Algunos de sus puntos más fuertes son la programación basada en comportamientos, la capacidad de movilidad de código y la gestión de la comunicación entre agentes.

Otro framework distribuido surgido en los últimos años, el cual sí que está completamente orientado al campo de la robótica es ROS [141]. Este framework cumple ampliamente los requisitos de las arquitecturas distribuidas ideales, es modular, robusto y flexible. De hecho, dentro de sus capacidades más destacadas se encuentra la de distribuir toda la carga computacional en distintos módulos software que se ejecutan independientemente pero se comunican frecuentemente entre sí para llevar a cabo algoritmos complejos. Sin embargo, una de las principales carencias de ROS es que está orientado a la programación distribuida de un solo robot, es decir, está pensado para que las comunicaciones sean locales entre procesos y no entre distintas plataformas robóticas, lo cual dificulta su aplicabilidad en casos de sistemas multirobot.

Uno de los principales retos que pretende superar este trabajo es conseguir el entendimiento y anexionamiento de estos dos frameworks con el objetivo de unificarlos en un único sistema compuesto por los beneficios que los dos frameworks individualmente ofrecen, para de esta manera aplicarlo sobre escenarios compartidos por sistemas robóticos heterogéneos, en este trabajo, orientado concretamente a lo que se conoce como sistemas de recursos y cómputo limitado.

El auge de las tecnologías en microcontroladores ha originado en la última década la aparición de un amplio surtido de procesadores embebidos en tarjetas ligeras portátiles de bajas prestaciones. Esta disminución de las dimensiones y el peso de las tarjetas las ha convertido en elementos imprescindibles a la hora de diseñar y construir nuevos robots orientados a la investigación o a aplicaciones en general no industriales. Además, cuando se trabaja con arquitecturas escalables y reutilizables, un buen resultado obtenido en una plataforma de recursos limitados ofrecerá los mismos o mejores resultados en plataformas de cómputo mayor, por lo que se podrán dedicar recursos a otras tareas requeridas.

En paralelo, la industria de las comunicaciones se encuentra en plena expansión y cada poco tiempo surgen nuevas versiones de protocolos de comunicaciones inalámbricas, cada vez más veloces y cada vez con menor consumo de energía y mayor número de nodos de conexión. Esto origina que los robots diseñados ofrezcan fácilmente algún tipo de comunicación inalámbrica lo que permite establecer una comunicación entre los robots de punto a punto sin necesidad de desarrollar una arquitectura de comunicaciones cliente-servidor como ocurría hace unos años. Esta comunicación punto a punto otorga mayor independencia si cabe a los robots y ofrece la posibilidad de que se realicen conversaciones, consultas y/o consensos entre robots lo cual dota al sistema de una mayor capacidad de cooperación y abre las puertas a la aplicabilidad de mecanismos de inteligencia artificial desarrollados sobre sistemas distribuidos.

El problema general a resolver dentro del campo de la robótica móvil se puede resumir en una única misión: *navegar*. La caracterización propia de un robot móvil es la calidad de cómo se desplaza de un punto a otro mediante ruedas. La navegación comprende una serie de complejas subtareas más o menos independientes sobre las cuales depende la calidad del resultado final. Uno de estos pilares que sustentan una buena navegación es la *localización*.

La localización es imprescindible dentro de las tareas de cualquier robot móvil ya que siempre debe determinar su posición y orientación en el espacio respecto a un sistema de referencia local o global para partir de un punto sobre el que desplazarse de manera controlada. Existen multitud de métodos en la literatura que afrontan este problema, de los cuales destacan los filtros de fusión por utilizar información de sensores embarcadas en el robot para mejorar la estimación de su posición y orientación. Sin embargo, en la mayoría de casos los tiempos de cómputo requeridos por estos métodos son muy costosos y no son aplicables a sistemas de recursos limitados.

Otra problemática principal de la navegación es la evasión de colisiones. Las estrategias de path planning sobre mapas conocidos lidian con este problema sobre obstáculos estáticos, sin embargo la complejidad del cálculo de la solución se dispara cuando se trabaja en un mismo escenario compartido por muchos robots donde cada uno desconoce la planificación de movimientos del resto.

Considerando todos estos factores, en este trabajo se pretende aprovechar las posibilidades ofrecidas por las arquitecturas distribuidas, sobre todo a nivel de comunicaciones, para ofrecer soluciones aplicadas a robots móviles de recursos limitados en sus problemáticas de localización, coordinación, detección y evasión de colisiones.

1.2 Justificación

Hoy en día existen multitud de plataformas robóticas con un alto nivel de prestaciones capaces de implementar costosos algoritmos en tiempo real. Sin embargo, el coste asociado a estos sistemas suele ser muy elevado y a la hora de experimentar con grupos de robots, el coste total del material resulta excesivamente alto. Por este motivo, en los campos de educación e investigación se suelen utilizar robots más accesibles y de bajo coste [167]. El ahorro en este tipo de sistemas empotrados va asociado a unas restricciones en capacidad de cómputo, tamaño de la memoria y calidad de prestaciones generales como el tipo de conectividad en las comunicaciones que ofrecen o la resolución de los sensores y actuadores incorporados. Estas bajas prestaciones impiden en muchos casos llevar a cabo la implementación de algoritmos complejos con una alta carga computacional, con una gran extensión o con altas necesidades de memoria, por lo que en algunos casos se utilizan computadores auxiliares para realizar dichos cálculos. No obstante, estos procedimientos no son coherentes con el concepto de un sistema autónomo e independiente. Por este motivo en este trabajo se lidia con el reto de que todos los algoritmos desarrollados sean implementables de manera local dentro de los robots, teniendo en cuenta sus limitaciones computacionales. Obtener unos buenos resultados en este tipo de sistemas escalables es extrapolable a su aplicación a sistemas de mejores prestaciones con el consiguiente ahorro de cómputo que supone, permitiendo la ejecución en paralelo de otras tareas.

Por otro lado, las soluciones ofrecidas a los problemas típicos de la robótica móvil, en general están enfocadas a su aplicabilidad en un robot individual el cual actúa según percibe el entorno mediante sus sensores. Sin embargo, cuando se trabaja con grupos de robots, la capacidad que tienen de comunicarse entre sí, puede añadir información nueva de entrada al sistema, abriendo un abanico de conocimientos relevantes que mediante su sensorización no son capaces de discernir ni percibir. Aprovechar convenientemente esta habilidad de comunicarse para obtener información adicional sobre el entorno y conseguir el beneficio del conjunto, ofrece nuevas soluciones y nuevas líneas de actuación totalmente compatibles con las ya definidas a nivel individual.

A nivel de software existen algunos frameworks distribuidos que ofrecen una modularidad y escalabilidad adecuada para sistemas robóticos y existen otros frameworks orientados hacía la arquitectura basada en agentes, los cuales están más orientados a la programación mediante comportamientos y la gestión de las comunicaciones entre agentes. La fusión de estas dos herramientas puede ofrecer una arquitectura distribuida que incorpore los beneficios que ambos frameworks presentan a nivel individual, resultando conveniente sobre todo en escenarios con múltiples robots heterogéneos.

Por estos motivos se considera necesaria la investigación sobre el desarrollo de una arquitectura distribuida basada en agentes aplicable a robots móviles de recursos limitados, sobre la que poder desarrollar nuevos algoritmos que ofrezcan nuevas soluciones a las problemáticas típicas del campo de la robótica móvil, teniendo en cuenta el beneficio de la capacidad de comunicación para otorgar cierta inteligencia al sistema y conseguir favorecer el desempeño tanto de las misiones individuales como del grupo completo de robots.

1.3 Objetivos

La presente tesis tiene como objetivo general desarrollar nuevas estrategias basadas en arquitecturas distribuidas para resolver las problemáticas relacionadas con la robótica móvil; concretamente en tareas de localización, coordinación, detección y evasión de colisiones, de modo que se aprovechen las ventajas comunicativas ofrecidas por este tipo de arquitecturas para ofrecer soluciones mejor coordinadas y más inteligentes. Para ello se proponen los siguientes objetivos específicos:

- Estudiar las arquitecturas distribuidas disponibles de manera libre y con soporte multiplataforma orientadas a la gestión de agentes y basadas en una programación modular y escalable.
- Realizar una revisión del estado del arte de la integración de arquitecturas distribuidas aplicadas a las problemáticas derivadas del control con robots móviles.
- Obtener un modelo de arquitectura aplicable a robots móviles de recursos limitados que permita ofrecer nuevas soluciones a las problemáticas propias de estas plataformas.
- Documentarse y experimentar con los distintos frameworks robóticos ofrecidos por los robots disponibles para realizar la integración de la arquitectura distribuida y las implementaciones de los algoritmos desarrollados.
- Estudiar los protocolos de comunicaciones ofrecidos por los distintos robots disponibles para conocer su gestión, velocidad y limitaciones.
- Desarrollar algoritmos basados en sistemas distribuidos que permitan llevar a cabo la autolocalización local y global de robots móviles de recursos limitados utilizando la información sensorial local y una fuente de localización global.
- Extender los algoritmos de autolocalización desarrollados a sistemas con grupos de robots heterogéneos de manera que se obtenga una metodología de localización cooperativa entre robots móviles.

- Aprovechar las ventajas de las comunicaciones entre agentes para ofrecer nuevas metodologías que hagan uso de la información intercambiada entre los mismos para adquirir más consciencia de ellos mismos y de su entorno, pudiendo actuar de una manera más precisa.
- Ofrecer algoritmos de navegación y coordinación basados en modelos distribuidos y aplicarlo en experimentos reales sobre las plataformas disponibles.
- Desarrollar nuevos algoritmos de detección y evasión de colisiones basados en el consenso y acuerdo a través de las comunicaciones entre plataformas móviles heterogéneas.
- Aplicar todo lo desarrollado a las plataformas reales disponibles para experimentación, realizando las pruebas pertinentes, recogiendo y analizando los resultados que deriven de las mismas.

1.4 Organización de la tesis

Con la delimitación realizada del tema de la tesis, se organizan los capítulos restantes de la siguiente forma:

Capítulo 2
Se presenta el estado del arte tecnológico y se definen los modelos de arquitectura y los frameworks robóticos propuestos. Además se exponen las primeras aportaciones de este trabajo al uso y la integración de los MAS con RSF mediante el desarrollo de middlewares de control.

Capítulo 3
Se contextualiza el estado del arte de la autolocalización de robots y se presenta un algoritmo de autolocalización local basado en filtros de fusión sensorial mediante corrección continua basado en el tiempo.

Capítulo 4
Se resumen los principales algoritmos de autolocalización global existentes y se propone un algoritmo de autolocalización global para robots móviles basado en métodos de correspondencia geométrica y aplicado a entornos de interiores.

Capítulo 5
Se establecen los algoritmos de autolocalización local y global basados en eventos para la mejora de la localización de robots móviles de recursos limitados, utilizando los principios de control y muestreo basados en eventos y tomando como punto de partida los algoritmos en cascada de corrección continua basados en el tiempo.

Capítulo 6

Partiendo del algoritmo desarrollado en el capítulo anterior, se presentan y se comparan dos algoritmos de localización cooperativa entre robots, uno que realiza una actualización continua por tiempo y otro basado en eventos.

Capítulo 7

Se describen los algoritmos de navegación y coordinación para sistemas distribuidos mediante el uso de la arquitectura multi-agente y se aplican sobre robots móviles reales.

Capítulo 8

Se presenta un breve resumen de los métodos existentes y se propone una metodología de detección y evasión de colisiones mediante acuerdos entre robots llevados a cabo a través de consensos entre agentes.

Capítulo 9

Se establecen las conclusiones obtenidas junto con las posibles líneas de continuación de la presente tesis en trabajos futuros.

Capítulo 2

Arquitecturas multi-agente y frameworks robóticos

Dentro del área de investigación de la robótica existe el interés generalizado de conseguir integrar los sistemas multi-agente (MAS) dentro de los frameworks de software orientados a la robótica (RSF) tal y como justifica se justifica en el presente capítulo. Un gran número de trabajos justifican el interés por unificar estos dos sistemas en una única arquitectura que combine sus funcionalidades, no obstante no existe una metodología específica a seguir para conseguirlo. Este capítulo se centra en exponer las aportaciones de la presente tesis a la necesidad de integración de los MAS con los RSF. En primer lugar se describe el estado del arte tecnológico de las arquitecturas distribuidas y seguidamente se detalla la arquitectura multi-agente utilizada para el desarrollo de demostradores tanto en simulación como en el desarrollo práctico con robots reales, JADE. Posteriormente se exponen las aportaciones realizadas sobre dicha plataforma para la mejora de su gestión y funcionalidad. A continuación se describen los frameworks robóticos de software utilizados en la mayoría de demostradores prácticos del presente trabajo, ROS y LeJOS, así como las posibilidades que ofrecen para su uso en sistemas multi-robots. Seguidamente se expone brevemente el software de simulación de robots V-REP así como la incorporación de nuevos modelos de robots al programa. Por último se expone la aportación principal de este capítulo: la integración del sistema multi-agente JADE a los frameworks robóticos ROS, LeJOS y el simulador V-REP para su aplicación en sistemas multi-robots heterogéneos y distribuidos.

2.1 Estado del arte tecnológico sobre arquitecturas distribuidas

Para el desarrollo de la presente tesis se ha llevado a cabo una amplia recopilación y revisión del estado del arte de las arquitecturas basadas en sistemas multi-agente así como de su aportación a distintas problemáticas relacionadas con el área de la robótica móvil en plataformas de recursos limitados. Esta sección recoge dichos antecedentes situados entre los campos de la Inteligencia Artificial y la Robótica con el objetivo de contextualizar el marco sobre el cual se sitúan las aportaciones de la presente investigación.

2.1.1 Introducción a los sistemas multi-agente

Los sistemas multi-agente, conocidos como MAS (Multi-Agent Systems), provienen de una rama del campo de la Inteligencia Artificial (IA) la cual intenta unificar en una sola disciplina los conceptos de la ingeniería del software y de sistemas distribuidos inteligentes. El concepto de sistema multi-agente surgió de manera teórica hace poco más de 30 años y causó tal impacto que, durante estos años la comunidad investigadora de la IA se ha esforzado por consolidar dicho concepto en una ingeniería de software de diseño denominada "Ingeniería de Software Orientado a Agentes", conocida por las siglas AOSE (Agent-Oriented Software Engineering). Esta metodología se considera una evolución de la programación orientada a objetos ya que ofrece las mismas propiedades de abstracción, modularidad y reutilización de código pero además añade habilidades de autonomía, comunicación e inteligencia. De hecho, en 2005 se publican trabajos como [153] donde ya se enfoca el concepto de IA específicamente al desarrollo íntegro de agentes. Para facilitar este desarrollo, definir dicha ingeniería y llevar su uso a experimentos prácticos, durante años se han desarrollado algunas metodologías como Gaia [196], INGENIAS [135] o GORMAS [10], también han surgido herramientas para el soporte del software de agentes como ADELFE [19], JACK [193], KAoS agent toolkit [26] o JADE [17] y al mismo tiempo han aparecido importantes organizaciones como la FIPA (Foundation for Intelligent Physical Agents) [131] , OMG Agent Standardization [132] o KQML agent communication [62] las cuales promueven la especificación de una metodología estandarizada para el desarrollo de plataformas multi-agente. Hoy en día, a pesar de dichos esfuerzos sigue sin existir una metodología estándar generalizada que marque unas pautas estrictas para los desarrolladores. No obstante, el auge que esta disciplina ha experimentado es bien reconocido en muy diversos campos de aplicación. El gráfico mostrado en la Figura 2.1 está extraído de [161], y refleja el porcentaje de madurez que tienen las soluciones basadas en agentes aplicadas a distintas temáticas clasificadas directamente según su dominio de aplicación, lo cual es conocido comúnmente por el concepto de sectores verticales. Hay que tener en cuenta, por ejemplo, que el sector de *Defensa* es difícil de estimar debido a su alto grado de confidencialidad.

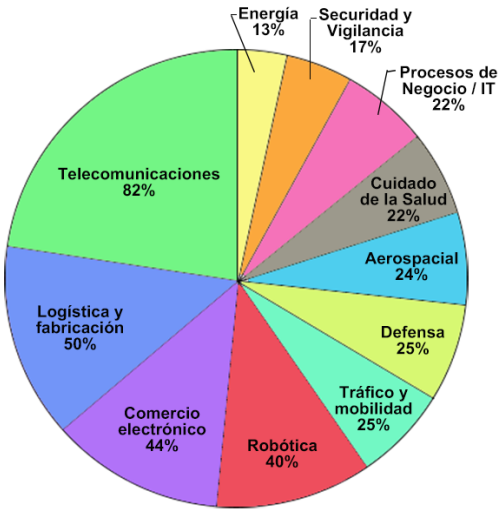


Figura 2.1: Madurez de las soluciones basadas en agentes para sectores verticales.

En el gráfico, el porcentaje de madurez se entiende como el nivel de integración e instauración de los MAS como sistemas que forman parte actualmente de procesos operacionales en entornos comerciales no controlados. Un nivel muy bajo de madurez significa que la tecnología de MAS se plantea tan sólo como un prototipo de solución en fase de investigación. Como se puede observar, el sector que mejor ha integrado el uso de soluciones basadas en agentes es el sector de las telecomunicaciones [7] con una madurez del 82 %. El sector de la robótica queda muy por debajo de esta cifra con un 40 % de madurez, situándose aún así en el cuarto puesto por debajo de logística y manufacturación [154] y el sector del comercio electrónico [136]. La idea de que este paradigma de programación consiga mayor madurez en el sector de la robótica ofreciendo nuevas soluciones surge directamente de la similitud entre el concepto de agente y el concepto de robot, la cual se detalla a continuación.

2.1.2 Modelo de Agente

Varios artículos de la literatura como [67], [128] o [195] han definido el término de agente software desde distintos puntos de vista ofreciendo diferentes matices según su ámbito de aplicación. Una de las definiciones más genéricas se encuentra en [153]: *Un agente es cualquier cosa capaz de percibir algo de su entorno a través de sensores y puede actuar en ese entorno a través de actuadores.* En la misma referencia se encuentra una representación formal del modelo de agente donde cabe describir en primer lugar la perspectiva del sistema sobre el cual se va a trabajar y en segundo lugar la arquitectura abstracta que define el modelo.

La especificación del sistema define el ámbito y el marco sobre el que los agentes trabajan. Esta taxonomía se caracteriza fundamentalmente por cuatro parámetros: $\{\mathcal{P},\mathcal{E},\mathcal{A},\mathcal{S}\}$. Donde $\mathcal{P}$ (*performance measure*) representa el conjunto de las percepciones que se pueden obtener del entorno. $\mathcal{E}$ (*environment*) define el escenario, el marco o el ámbito sobre el cual se aplica el sistema. $\mathcal{A}$ (*actuators*) corresponde al conjunto de acciones que modificarán o actuarán sobre el entorno de alguna forma y $\mathcal{S}$ (*sensors*) reúne los sensores o los canales mediante los cuales es posible percibir cambios en el entorno.

Estos parámetros crean un contexto sobre el cual es posible definir una arquitectura apropiada para un agente que desee involucrarse dentro del sistema. La arquitectura estándar (Arq_s) del modelo de agente se define por la tupla:

$$Arq_s =< S, A, acción, entorno > \tag{2.1}$$

Donde S se corresponde con un conjunto finito ($\{s_1,s_2,...,s_n\}$) de la descripción de todos los posibles estados del entorno. A representa el conjunto ($\{a_1,a_2,...,a_n\}$) de todas las acciones que el agente es capaz de llevar a cabo. El parámetro *acción* es la función que describe el comportamiento de los agentes de dicha arquitectura y *entorno* corresponde a una función que describe el comportamiento dinámico que puede tener el entorno. De este modo, el comportamiento o la *acción* de un agente viene definido por la función:

$$acción : S^* \implies A \tag{2.2}$$

S^* contiene un conjunto o subconjunto secuencial de estados donde el orden importa y por tanto implica que una acción deriva de una secuencia de cambios de estado, sin embargo este formalismo no representa específicamente un propósito a priori o una planificación prevista en el orden en el que se ejecutan dichas acciones, tal y como se demuestra en [153].

Por otro lado, el resultado de aplicar sobre un *entorno* en un estado s_j, una acción a_k, genera como resultado un conjunto de nuevos estados S'. Esto se puede representar como $entorno(s_j, a_k) = S'$, o de manera genérica, formalizar como:

$$entorno : SxA \implies S' \tag{2.3}$$

Como puede observarse en la descripción del modelo y las arquitecturas, existe un alto grado de abstracción en los conceptos que definen un agente software. Además, la mayoría de las nomenclaturas que se utilizan, hacen referencia implícitamente a elementos hardware físicos comúnmente conocidos dentro del área de la robótica como son actuadores y/o sensores. Por este motivo no es de extrañar que los conceptos de *agente software* y de *robot*, frecuentemente se relacionen e incluso se citen de manera similar dentro de este campo. No obstante, con el tiempo han surgido distintas arquitecturas o frameworks robóticos especializados en la

aplicación de los MAS al campo de la robótica, tal y como se describe en el siguiente apartado.

2.1.3 Sistemas Multi-agente Robóticos

La comunidad de investigadores del campo de la robótica reflejó un gran interés por los MAS desde el momento de su aparición, ya que dicha disciplina ofrecía un nuevo punto de vista prometedor para la solución de algunas problemáticas propias de este campo. De hecho, no tardaron en detallar unas primeras especificaciones sobre los requisitos que debía cumplir un sistema robótico basado en dicha arquitectura, extraídas de [133], [92], [60], [31]:

- *Concurrencia y arquitectura distribuida.* Se debe poder utilizar los recursos hardware de una manera concurrente e individualizada, para poder aprovechar al máximo las prestaciones computacionales ofrecidas por el sistema.
- *Modularidad.* Cada componente del sistema debe estar individualizado a nivel de software y al mismo tiempo guardar cierta relación de orden entre el resto de componentes. Estas dependencias entre componentes deben mantenerse en un mínimo con el fin de que la arquitectura sea escalable y reutilizable o adaptable a cambios o mejoras.
- *Robustez y tolerancia a fallos.* El mal funcionamiento de un componente no puede bloquear todo el sistema, es más, lo deseable es que el sistema sea capaz de seguir funcionando de la mejor manera posible con los recursos de los que dispone para conseguir sus objetivos. Para ello, el software que representa esos componentes hardware (agentes), debe ser capaz de actuar por iniciativa propia y tomar decisiones para superar dichas situaciones.
- *Tiempo real y eficiencia.* Cuando se trabaja con sistemas robóticos, normalmente existen restricciones de tiempo real que resultan complicadas llevar a cabo en sistemas distribuidos. La eficiencia resulta crucial sobre todo en sistemas de recursos computacionales limitados para garantizar el cumplimiento de las restricciones de tiempo real.

La integración de estas especificaciones propias de los MAS al campo de la robótica ha desembocado en una nueva terminología para estos sistemas conocida como MARS (Multi-Agent Robotic System), la cual todavía hoy en día continúa suponiendo un reto sin una metodología estipulada a seguir para conseguirlo. Los MARS tienen como objetivo ofrecer una arquitectura resultante más reutilizable, escalable, flexible, robusta y modular, que las arquitecturas robóticas tradicionales.

Los frameworks o arquitecturas para software orientados a robótica o RSFs (Robotics Software Frameworks) han intentado durante los últimos años ofrecer un conjunto de herramientas, librerías, algoritmos y controladores, genéricos con la

finalidad de poder ser utilizados en cualquier tipo de sistema robótico evitando así reinventar constantemente la rueda como ocurría hace años, donde cada sistema hardware normalmente integraba su propio software cerrado y cambiar de sistema suponía cambiar de método, lenguaje y diseño de programación. Además, esta evolución de los RSFs ha estado orientada también hacia el software libre y hacia comunidades de desarrollo abiertas donde cualquier investigador puede participar y colaborar. En diversos trabajos estos frameworks también son conocidos como Middlewares robóticos (Robotics Middleware) o software de sistemas robóticos (Robotics Software Systems).

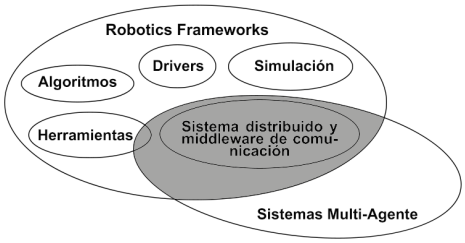


Figura 2.2: Relación conceptual entre los MAS y los RSFs extraída de [83].

Precisamente esta evolución junto con la aparición de los MAS, ha hecho que hoy en día, los dos sistemas ofrezcan herramientas y soluciones similares en muchos aspectos, sobre todo en la gestión de las comunicaciones dentro de la arquitectura distribuida. La figura 2.2 muestra sombreado este solapamiento entre los dos conceptos. Algunos RSFs ya integran funcionalidades de los MAS como métodos de intercambio de mensajes entre arquitecturas, servicios de páginas amarillas, paso de mensajes P2P,... Por lo que ya estos RSFs presentan una infraestructura inicial adecuada para el desarrollo de MARS. Entre los RSFs más utilizados cabe destacar los siguientes por ser opensource y por ofrecer una buena infraestructura de middleware distribuido para poder representar un MARS, tal y como se muestra en [83]: OpenRDK [36], OpenRTM [6], OROCOS [32], ROS [141] o YARP [120].

Una vez planteados los RSFs adecuados para su integración con MAS, cabe destacar que existen dos niveles de aplicación a nivel conceptual, donde los MAS son integrados en los RSFs. El nivel más alto (figura izquierda de la figura 2.3) se aplica cuando se mantiene una correlación directa entre el agente software y el robot, es decir, cada robot representa un agente individual dentro del sistema y comparte el entorno total o parcialmente con el resto de agentes o robots. Este nivel es el más común ya que normalmente se asocia el concepto de robot a la representación del modelo físico de un agente software, aunque no siempre tiene por qué ocurrir así. En un nivel de aplicación más bajo (figura derecha de la figura 2.3), los MAS se utilizan también para gestionar los distintos dispositivos individuales que componen un robot e incluso las distintas funcionalidades que puede ofrecer.

Esto quiere decir que en el caso hipotético de trabajar con un robot móvil, es posible que un agente se encargue del movimiento de las ruedas, otro de obtener las imágenes obtenidas de una cámara embarcada, otro de interpretarlas, otro de recoger las medidas que devuelve un láser, etc. De este modo, se crea un sistema distribuido donde la correcta coordinación entre estos agentes consigue llevar a cabo las distintas funcionalidades que puede ofrecer el robot.

Figura 2.3: A la izquierda se muestra una arquitectura MARS aplicada a alto nivel donde cada agente representa un robot y a la derecha se muestra un ejemplo de una arquitectura MARS aplicada a bajo nivel donde cada agente representa un componente o una funcionalidad.

Estas dos estrategias no son excluyentes por lo que en ocasiones se combinan dando lugar a MARS compuestos por robots o agentes que están gobernados internamente por otros MARS más simples. Este concepto de obtener sistemas complejos a partir de subsistemas estables más sencillos se conoce como teoría de Sistemas Holónicos.

2.1.4 Sistemas Holónicos

El término *holón* aparece por primera vez en el libro *The Ghost in the Machine* [90] escrito por Arthur Koestler. Se define holón como una entidad que es a la vez un todo y una parte. En un trabajo posterior, [12], Koestler define un Sistema Holónico (en adelante, SH) como una organización altamente distribuida donde la inteligencia se distribuye entre entidades individuales. Un SH está compuesto por un conjunto de entidades autónomas (holones) que cooperan entre sí para lograr un objetivo común. El holón tiene un comportamiento autónomo que es implementado mediante un componente inteligente deliberativo con capacidad de comunicación y de reacción ante estímulos o eventos en el entorno (o mensajes de otros holones); y puede tener, por ejemplo, un componente físico (máquina, robot, herramienta) que es el elemento que controla como ocurre en [45]. Los SH han sido utilizados con éxito en el área de sistemas de fabricación inteligente (IMS) [76] y en otras áreas de computación inteligente distribuida. Tal y como se puede observar, no resulta sencilla la distinción a nivel conceptual de las definiciones de

agente y holón por lo que ambos términos comparten protagonismo en diversos trabajos como [187], [63] o [33]. En [1] por ejemplo, se recogen una serie de trabajos basados en arquitecturas de holones donde se emplean agentes como componentes software principales. Algunos investigadores como Bussmann ([172]) o Brennan y Norrie ([28]) han intentado aclarar las diferencias filosóficas fundamentales entre estos dos conceptos. Tal y conforme se sintetiza en [1] y en [97], la diferencia más destacable reside en la interpretación de la información y de su procesamiento físico. En el caso del holón, esta separación entre el software y el hardware es explícita y requerida, mientras que en el agente no existe una separación explícita como tal en esta interpretación. La figura 2.4 extraída de [97], compara las entidades de holón y agente en cuanto a los conceptos teóricos de autonomía y cooperatividad.

Desde la perspectiva de un MAS, un SH es una especialización de un MAS, donde cada holón es precisamente un agente. Es por este motivo por lo que surge una terminología unificada para referirse a los sistemas que combinan los dos conceptos bajo una misma estrategia: Sistemas Holónicos Multi-agente (conocido como Holonic Multi-agent Systems), tal y como se referencian en los siguientes artículos [182, 158, 79, 148, 114].

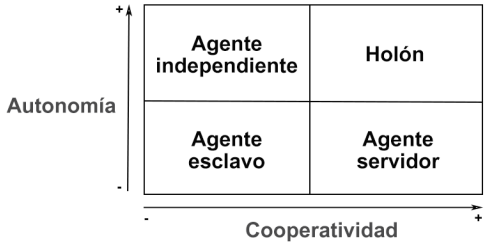


Figura 2.4: Comparativa conceptual teórica sobre los conceptos de agente y holón donde se aprecia visualmente que un holón es una especificación de un agente con alto grado de autonomía y de cooperatividad.

Normalmente, por motivos de eficiencia se utiliza un agente individual para representar un holón, sin embargo también es posible realizar esta representación mediante la agrupación selectiva de varios agentes. En cualquier caso, entre éstos suele existir un "mediador"(según se especifica en [162]) o *Head* quien representa las intenciones del holón y negocia dichas intenciones con los agentes o holones del entorno. La unión entre este agente-representante y el resto de agentes que forman parte del *cuerpo* del holón suelen definirse mediante especificaciones o compromisos (*Commitments*) las cuales no tienen porqué ser estáticas o preconcebidas.

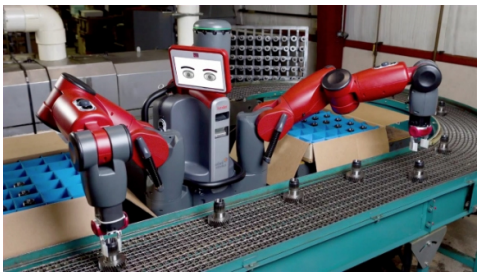


Figura 2.5: Colaboración de dos brazos robóticos en entornos industriales. Robot Baxter desarrollado por Rethink Robotics. (`/www.rethinkrobotics.com`)

Un Sistema Holónico Multi-agente con un conjunto de agentes A_t y un conjunto de holones H_t para un tiempo t, se puede definir recursivamente de la siguiente manera:

- para cada $a \in A_t$, $h = (\{a\}, \{a\}, \emptyset) \in H$, donde cada agente presente en el sistema formaría parte de un holón independiente o *atómico*, y
- $h = (Head, SubHolones, C) \in H$, donde $SubHolones \in 2^H \backslash \emptyset$ corresponde al conjunto de holones que participan en h. $Head \subseteq SubHolones$ es el conjunto no vacío de holones que representan el holón en el entorno y son responsables de la coordinación de las acciones en el interior del holón. $C \subseteq Commitments$ define los compromisos acordados y aceptados entre todos los holones $h' \in SubHolones$ y el $Head$ en el momento en el que formaban parte de h.

Un holón h puede ser observado por el resto como cualquier otro agente del sistema en A_t. Sin embargo es posible que también se reconozca como el conjunto de agentes que lo componen o lo representan. Como cada $Head$ posee una identificación única es posible comunicarse únicamente con él a través de su dirección. Dado el holón $h = (Head, \{h_1, h_2, ..., h_n\}, C)$, donde $h_1, h_2, ..., h_n$ son *subholones* de h y h es el *superholón* de $h_1, h_2, ..., h_n$, el conjunto $Body = h_1, h_2, ..., h_n \backslash Head$ representa los *subholones* que no pueden representar al holón h, es decir no pueden formar parte del $Head$. A su vez, los *subholones* h' pueden formar parte de otros holones al mismo tiempo siempre y cuando esto no contradiga ninguno de los compromisos con los *superholones* o *subholones*. Por otro lado C especifica la estructura de la organización tal y conforme se explica en [157].

Existen distintos trabajos donde se relacionan y se aplican los SH en sistemas robóticos integrados dentro de sistemas multi-agente, como [84], [178] o [24]. La mayoría de estos trabajos están orientados hacia la industria de la fabricación automatizada mediante robots o máquinas industriales (figura 2.5) donde los cambios en el entorno están ligeramente acotados. Sin embargo, el nivel de dificultad

aumenta para el caso del desarrollo de la robótica móvil en entornos no controlados. Se encuentran algunos trabajos relacionados sobre todo con la coordinación (como los robots constructores inspirados en termitas, figura 2.6(a), desarrollados por la universidad de Harvard), la colaboración de grupos de robots (la figura 2.6(b) muestra 5 s-bots colaborando entre ellos) o como casos de agrupaciones en colonias como en [118] o competiciones entre equipos de robots como en [37] o [46]. No obstante, cada uno de ellos sigue su propia metodología no-estandarizada para alcanzar sus objetivos.

(a) Robot constructor TERMES. (b) Robots móviles S-bot

Figura 2.6: Coordinación y colaboración multi-robot con distintas finalidades.

En general, la mayoría de trabajos de la literatura relacionados con la robótica utilizan el término general MAS para denotar la metodología utilizada con la intención de ofrecer una generalización la cual puede incluir en algunos casos sistemas holónicos integrados.

2.1.5 Sistemas Distribuidos Aplicados a Robots Autónomos

Las aplicaciones sobre colonias de robots heterogéneos o misiones colaborativas entre distintos grupos de robots, suelen ser la principal aplicación de los MAS aplicados al campo de la robótica móvil. Existen multitud de grupos de investigación a nivel internacional trabajando sobre esta línea como el Departamento de Informática dirigido por Frederick P. Brooks Jr. de la universidad de Carolina del Norte en Chapel Hill, Estados Unidos (`/http://gamma.cs.unc.edu/`); el Laboratorio de Sistemas Autónomos del Instituto de Robótica y Sistemas Inteligentes de la Universidad de Zurich, Suiza (`/http://www.asl.ethz.ch/`); el Laboratorio de computación ubicua y sistemas multi-agente móviles del Departamento de Informática de la Universidad de Saskatoon, Canadá (`/http://madmuc.usask.ca/`); o el Laboratorio de Sistemas Robóticos e Inteligentes del Instituto Tecnológico de Georgia, en Atlanta, Estados Unidos(`/http://www.robotics.gatech.edu/`). A nivel nacional también existen destacados grupos como el grupo de RoboticsLab de la Universidad Carlos III

de Madrid (/http://roboticslab.uc3m.es/roboticslab/); o el grupo Systems, robotics & vision de la Universitat de les Illes Balears (/http://srv.uib.es/).

Cada grupo utiliza diversas metodologías para estas investigaciones con agentes, sin embargo existe un planteamiento conceptual inicial compartido y generalizado entre los distintos grupos.

Hay que abordar estos problemas de colaboración de una manera individualizada sobre cada agente o robot, pero al mismo tiempo, el resultado de estas acciones individuales o locales deben ser capaces de ofrecer una solución global desde el punto de vista del conjunto completo o la comunidad de agentes que están interviniendo en ese momento. (Magnus Egerstedt, Colloquium at Halmstad University, 2014.)

Por ejemplo, suponiendo un escenario con N robots, si se pretende que se organicen de manera que entre todos se sitúen formando un círculo, un planteamiento erróneo sería establecer una decisión centralizada con un sistema supervisor o dictatorial el cual le comunicase a cada robot la posición sobre la que debe situarse para formar esa determinada forma geométrica entre todos. Lo que se pretende es que cada agente, de manera individual y normalmente sin necesidad del uso de comunicaciones, sea capaz de saber situarse en una posición que considere adecuada, únicamente observando su entorno y la posición de sus vecinos. De ese modo, en el momento en el que todos están quietos y convencidos de que están en la posición correcta, el sistema converge y entre todos forman un círculo.

En resumen, este tipo de enfoque se centra en las observaciones que lleva a cabo cada agente sobre el entorno y sus vecinos para llevar a cabo el control individual de cada uno. Uno de los grupos de investigación más importantes que hacen uso de este enfoque es el GRITSLab, de la Universidad de Georgia, Atlanta, (/http://gritslab.gatech.edu/}) dirigido por el Dr. Magnus Egerstedt.

Debido a la actual emersión de las redes de sensores, los equipos de robots a gran escala y las redes de dispositivos embebidos, es necesario desarrollar algoritmos interconectados con escasa comunicación, computación y con capacidades de detección. M. Egerstedt, Aug 2013.

Estas investigaciones hacen uso de la teoría de grafos para crear esquemas basados en nodos y aristas de proximidad que representan una abstracción del problema real como se explica en [119] y [168].

En la práctica, el uso de esta estrategia requiere una capacidad de detección del entorno y de los agentes vecinos muy fiable, ya que su convergencia depende de ello de manera directa. Para dispositivos de recursos limitados, ésto supone una dificultad añadida al problema, sobre todo en entornos dinámicos no controlados, ya que este tipo de material con el que se suele trabajar para realizar los demostradores prácticos, no suele tener una instrumentación precisa o fiable, introduciendo en el sistema una complejidad añadida independiente a la estrategia utilizada que

se desea demostrar. Por ejemplo, en un caso práctico, mediante infrarrojos un robot puede detectar un objeto pero para saber si ese objeto se corresponde con otro robot o con cualquier otra cosa es necesario aplicar alguna estrategia de identificación o *matching*, lo cual puede complicar el experimento. Este tipo de problemas es habitual en investigación dado el uso generalizado de sistemas embebidos de recursos limitados ya que este tipo de material resulta económico y dentro de sus limitaciones puede ofrecer un desempeño funcional parecido al de sistemas profesionales o industriales.

2.1.6 Sistemas de recursos limitados

Tiempo atrás, el material robótico para realizar experimentos prácticos que era posible encontrar en el mercado, resultaba altamente costoso. Además, normalmente cada fabricante ofrecía una plataforma cerrada con su propia API de programación e incompatible con nuevas versiones del producto o con otras plataformas. Del mismo modo que ocurre con el desarrollo software, en los últimos años ha surgido la tendencia generalizada de desarrollar hardware libre. La producción de tarjetas microcontroladoras de hardware libre se ha disparado en la actualidad originando un amplio abanico de productos de bajo coste que cada vez resultan más potentes computacionalmente pero sin llegar a ofrecer prestaciones profesionales.

La tabla 2.1 muestra una comparación entre las tarjetas microcontroladoras que más auge han tenido en el último año y con las que para algún caso se ha trabajado en la presente tesis con el fin de desarrollar demostradores.

						
Board:	MintDuino	Arduino Uno	Arduino Due	Netduino 2	Raspberry Pi2	BeagleBone Black
Price:	$24.99	$29.99	$49.99	$34.99	$39.99	$45.00
Processor:	ATmega328	ATmega328	ARM Cortex-M3	STMicro Cortex-M3	ARMCortex-A7	ARM Cortex-A8
Processor Speed:	16 MHz	16 MHz	84 MHz	120 MHz	900 MHz	1 GHz
Analog Pins:	6	6	12	6 (12-Bit)	-	7
Digital Pins:	14 (6 PWM)	14 (6 PWM)	54 (12 PWM)	22 GPIO (6 PWM)	8 Digital GPIO	65 GPIO (8 PWM)
Memory:	SRAM 2KB EEPROM 1KB	SRAM 2KB EEPROM 1KB	SRAM 96 KB	Code 192KB RAM 60KB	RAM 1GB	DRAM 512MB DDR3L, eMMC 2GB
Language:	Arduino C Variant	Arduino C Variant	Arduino C Variant	Microsoft C# or Visual Basic	Any	Any

Tabla 2.1: Comparación de tarjetas embebidas actuales

A pesar de que estos sistemas estén limitados computacionalmente, la mayoría de ellos incorpora un sistema de comunicaciones de alta tasa de datos de envío y recepción como protocolos Wi-Fi, Ethernet, Bluetooth o Zigbee. Por lo que es posible beneficiarse de esta capacidad de comunicación para intercomunicar dichos sistemas y poder ofrecer mejores resultados globales.

2.1.7 Gestión de Comunicaciones

La metodología descrita en la sección 2.1.5 intenta evitar al máximo el uso o la dependencia de las comunicaciones entre agentes para su convergencia ya que está inspirada en colonias o agrupaciones de animales donde no se ha demostrado que exista comunicación no-visual entre ellos y considera que su uso puede repercutir en problemas de retardos, saturación del ancho de banda de los canales de la comunicación o malas interpretaciones en negociaciones o acuerdos. De hecho, este tipo de problemas relacionados con las negociaciones o acuerdos mediante el establecimiento de comunicaciones, suelen ser objeto de estudio en otros campos menos relacionados con la robótica móvil, como sistemas de comercio electrónico [174], procesos de negocio [134] o en teoría de juegos [122].

Hoy en día, si se compara el rango de funcionamiento de la mayoría de los sensores embarcados en sistemas robóticos con el radio de cobertura que son capaces de alcanzar los sistemas de comunicaciones que generalmente llevan integrados, sin duda los sensores de detección física suelen estar mucho más limitados en distancia que los rangos ofrecidos por las comunicaciones. Es decir, que dado un grupo de robots G_R de N miembros, donde cada robot $R_i \in G_R$ con i =1,...,N y $N \geq 2$, tiene un rango de detección D_r y un rango de cobertura de las comunicaciones C_r, se cumple que $D_r \leq C_r$ para la mayoría de sistemas robóticos existentes en la actualidad; es decir, son capaces de establecer antes una comunicación inalámbrica con cualquier otro sistema, que detectar una posible presencia dentro del rango de detección de los sensores. Esto se acentúa aún más cuando se trabaja con sistemas de recursos limitados, como a menudo ocurre en el campo de la investigación, cuyos sensores suelen ser de bajo coste y por tanto de menor alcance. En este tipo de sistemas la correcta gestión de las comunicaciones puede jugar un papel fundamental ya que puede suponer una entrada nueva de información al sistema que facilite o acelere su convergencia. De este modo, sería posible realizar un control del sistema por encima de los anteriormente mencionados, previo e incluso complementario a las estrategias más utilizadas hoy en día. Precisamente éste es el marco sobre el cual se desarrollan las aportaciones de la presente tesis.

2.2 Arquitectura Multi-agente: JADE (Java Agent DEvelopment Framework)

2.2.1 Introducción

Entre la gran variedad de plataformas multi-agente surgidas en los últimos años, JADE aún se posiciona hoy en día como una de las mejores opciones según distintos estudios y trabajos como [101], [75] o [83]. Las principales razones por las que JADE destaca entre el resto es por haber mantenido su soporte y mantenimiento

desde sus inicios, por continuar en constante desarrollo y evolución gracias a un equipo de desarrolladores activo, por ser un sistema multiplataforma sin ninguna exclusividad respecto al sistema operativo sobre el que funciona y por poseer una amplia y activa comunidad de usuarios.

JADE surgió como una plataforma software de programación de agentes implementada estrictamente en Java en julio de 1998 y desde entonces ha estado en continuo desarrollo llegando a estar hoy en día en su versión 4.5.0 publicada el 8 de junio de 2017. JADE fue desarrollado originalmente por la compañía de telecomunicaciones Telecom Italia y se empezó a distribuir como software libre y código abierto a partir de su versión 1.3 disponible desde febrero de 2000 desde su sitio web `/http://jade.tilab.com/`. En marzo de 2003 Telecom Italia junto con Motorola crearon la organización *JADE Governing Board* con el objetivo de promover la evolución y la instauración de JADE en la industria de las telecomunicaciones móviles a modo de middleware.

Uno de sus logros más destacable es que desde sus inicios, JADE ha seguido el modelo de referencia especificado por el consorcio internacional FIPA (Foundation for Intelligent Physical Agents). Esta organización aglutina tanto instituciones académicas como compañías con interés en la tecnología de agentes, y es la encargada de producir unas especificaciones para la interacción entre agentes y sistemas heterogéneos. La especificación FIPA define por ejemplo, cómo dos agentes deben ser capaces de localizarse y comunicarse entre sí por medio de algunos mecanismos que deben existir dentro del sistema, como el registro de agentes o el intercambio de mensajes entre ellos, pero sin concretar unos protocolos de implementación específicos. El propósito final de dicha organización es proponer la especificación de una arquitectura independiente del protocolo o sistema de transporte de mensajes que se utilice, del lenguaje de comunicación usado entre los agentes, del sistema de gestión de agentes y del servicio de directorios que se utilice, tal y como se expone en la figura 2.7. De este modo, cualquier agente que esté definido o implementado conforme al mismo estándar, puede actuar conjuntamente con otros agentes y otras plataformas. JADE soporta la coordinación de múltiples agentes FIPA y proporciona una implementación estándar del lenguaje de comunicación de agentes FIPA-ACL definido en [104] por la misma organización. En el apéndice A se describen los elementos obligatorios que debe tener una arquitectura abstracta basada en la especificación FIPA.

JADE es completamente compatible con Java Development Kit (JDK) 1.4 o superiores e incluye tres elementos fundamentales recogidos en [18]:

1. Una librería formada por clases Java que proporcionan componentes y funciones para dar soporte a los desarrolladores en la creación de agentes, así como en el manejo de información mediante el uso de ontologías y la implementación de la especificación FIPA-ACL para el envío y la recepción de mensajes entre los agentes mediante protocolos estandarizados.

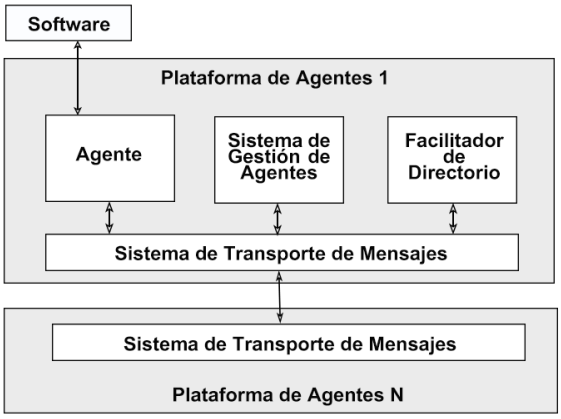


Figura 2.7: Esquema del modelo de referencia definido por FIPA.

2. Un entorno gráfico de gestión de la plataforma multi-agente en tiempo real desde donde es posible la creación o eliminación de los agentes, su agrupación en distintos contenedores o plataformas, o el envío de mensajes a agentes existentes.

3. Una interfaz gráfica con herramientas que permiten monitorizar y registrar de forma visual las actividades de los agentes activos en el sistema en tiempo real y así facilitar la verificación del correcto funcionamiento del sistema.

Tal y como se especifica en [34] y se muestra en la figura 2.8, cada plataforma JADE debe disponer de un *contenedor principal* dentro del cual el resto de contenedores deben estar registrados. En una misma plataforma pueden haber registrados multitud de contenedores. Cada *contenedor* se corresponde con una instancia del entorno de ejecución de JADE. En la figura 2.8 se muestra un ejemplo de interconexión entre dos plataformas comunicadas mediante algún tipo de red de comunicación. La plataforma 1 está distribuida entre tres computadores (*Host 1, Host 2, Host 3*). El *contenedor principal* de la plataforma 1 se ejecuta en el *Host 1* y tanto el *Host 2* como el *Host 3* se registran en dicho contenedor al inicializarse. El contenedor principal de la plataforma 2 se ejecuta en el *Host 4* y en cada *Host* se ejecutan ciertos agentes *A1, A2, A3, A4, A5* que son capaces de detectarse y comunicarse entre sí.

Como se puede observar en la figura anterior, el *contenedor principal* siempre ejecuta dos agentes específicos que se crean de manera automática cuando se lanza la plataforma, de acuerdo con la especificación FIPA:

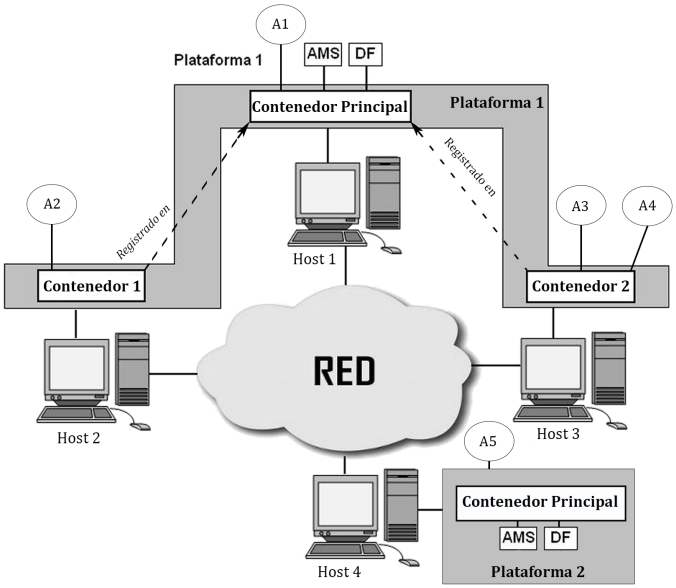


Figura 2.8: Esquema de las plataformas y los contenedores en JADE ([34]).

- AMS(Agent Management System): El cual proporciona un servicio de páginas blancas donde cada agente ejecutado dentro de la plataforma se encuentra registrado mediante un identificador único. El agente AMS supervisa la creación y destrucción de agentes, el acceso a la plataforma y a su uso, coordina solicitudes de otras plataformas y ofrece funcionalidades como la posibilidad de realizar búsquedas de agentes individuales por nombre.
- DF (Directory Facilitator). Este agente crea un directorio de páginas amarillas donde cualquier agente puede registrar y publicar el/los servicio/s que ofrece de tal forma que otro agente que esté interesado en solicitarlos, pueda identificar rápidamente quién es el agente que lo ofrece y ponerse en contacto con él. Cada plataforma dispone de un agente DF por defecto, sin embargo es posible lanzar más agentes de tipo DF con el fin de facilitar un catálogo de páginas amarillas distribuido por toda la red. Esta funcionalidad aporta al MAS características propias de la programación orientada a servicios (SOA: Service-Oriented Architecture).

De este modo, JADE proporciona las funcionalidades necesarias para el desarrollo de aplicaciones distribuidas peer-to-peer en entornos tanto estáticos como dinámicos. Los agentes están identificados con un nombre único dentro de la plataforma y son capaces de descubrir dinámicamente a otros agentes y comunicarse con ellos

conforme al paradigma punto a punto. Cuando arranca JADE se crea un canal de comunicación entre agentes llamado ACC (*Agent Communication Channel*) el cual controla el intercambio de mensajes entre agentes. Estos mensajes suelen ser asíncronos, lo cual permite el desarrollo de un modelo de comunicación distribuido y débilmente acoplado. Cada agente tiene una pequeña pila de entrada de mensajes donde el runtime de JADE almacena los mensajes enviados por otros agentes. Así mismo, cada vez que un mensaje es añadido a la cola de mensajes, el agente que lo recibe es notificado. Sin embargo, el instante en el que el robot recoge el mensaje de la cola y lo procesa depende únicamente del programador. En este sentido y siguiendo la especificación FIPA, es posible distinguir entre tres métodos distintos de definición del contenido de los mensajes entre agentes:

1. El método más simple consiste en el uso de cadenas de texto (*Strings*) para representar el contenido del mensaje. Este tipo de mensajes es conveniente cuando se intercambian contenidos con una semántica de tipo atómico como cifras simples, confirmaciones mediante booleanos, etc. No resulta adecuado hacer uso de cadenas de texto cuando el contenido de los mensajes contiene una semántica compleja, o conceptos abstractos, objetos u otros datos estructurados, ya que el coste y la complejidad de interpretación por parte del receptor del mensaje puede resultar tedioso y existen otros métodos más adecuados que ofrecen mayores facilidades para ello.

2. Un escalón más por encima del uso de las cadenas de texto se encuentra la metodología de utilizar objetos Java *serializables* para especificar el contenido de los mensajes. Los objetos *serializables* de Java ofrecen la ventaja de codificar el contenido de una clase completa con clases, métodos y variables, en una única instancia de la misma para enviar la clase entera en un sólo mensaje. El receptor, al recibir el mensaje, tan sólo debe realizar un *cast* al tipo de la clase que mandó el emisor para obtener una instancia idéntica a la que se envió. El problema de esta metodología radica en el hecho de que emisor y receptor deben compartir la definición de la clase de la instancia que intercambian. Además este método obliga a que los participantes en la conversación estén implementados necesariamente en Java.

3. El método más adecuado a utilizar cuando en el contenido de los mensajes existe una fuerte semántica asociada a los datos o información que se transmite es el uso de *ontologías*. Una ontología se puede definir como la especificación de una conceptualización o la descripción de los conceptos que pueden formar parte del conocimiento de un agente o una sociedad de agentes, incluyendo la información semántica que relaciona o envuelve dichos conceptos. Una ontología requiere que los objetos que se desean transmitir sean una extensión de las clases de ontologías predefinidas por JADE. De esta manera, cualquier plataforma es capaz de codificar y decodificar los mensajes en un formato estándar definido por la FIPA. JADE, divide las

ontologías en dos partes: un vocabulario el cual describe la terminología de conceptos utilizados por los agentes en sus espacios de comunicación y otra parte que describe la nomenclatura de las relaciones entre dichos conceptos, y que define su semántica y su estructura.

Para llevar a cabo la implementación de una ontología es necesario extender de la clase *Ontology* predefinida en JADE y añadir el conjunto de elementos que describen la estructura de los conceptos, acciones y predicados permitidos en el contenido de los mensajes. Esta metodología resulta la más adecuada sobre todo cuando se pretende comunicar JADE con otras plataformas multi-agente que también sean fieles a la especificación FIPA.

Cada agente en JADE se implementa como un único hilo de ejecución que debe heredar de la clase *jade.core.Agent* y sobrescribir los métodos *setup()* (encargado de la inicialización) y *takeDown()* (encargado de la finalización del agente) como mínimo.

Un agente está sujeto a un ciclo de vida en el que se definen los estados en los que se puede encontrar y sus posibles transiciones entre estados. El estado de un agente puede ser:

- *Iniciado:* Cuando el agente se ha creado pero todavía no se ha registrado en el AMS, no posee un nombre ni una dirección por lo que no le es posible comunicarse con otros agentes.
- *Activo:* Sucede cuando el agente se registra en el AMS y ya puede acceder a todas las funcionalidades de JADE.
- *Suspendido:* Cuando el agente está parado y su hilo de ejecución se encuentra detenido sin ejecutar ningún comportamiento.
- *En espera:* Cuando el agente se encuentra a la espera de que ocurra algún evento que despierte su ejecución, como que se cumpla cierta condición o se reciba un tipo de mensaje específico.
- *Desconocido:* Cuando se termina el hilo de ejecución, se elimina el agente del registro del AMS y termina.
- *Tránsito:* Cuando un agente va a migrar a una nueva localización, previamente entra en este estado en el que guarda sus mensajes en el buffer hasta que la migración se complete y vuelva al estado de *Activo*.

Las transiciones entre estos estados se llevan a cabo a través de métodos disponibles en la misma clase *jade.core.Agent*. La figura 2.9 muestra la relación entre los estados de los agentes y los métodos que corresponden a las transiciones de cambio de estado:

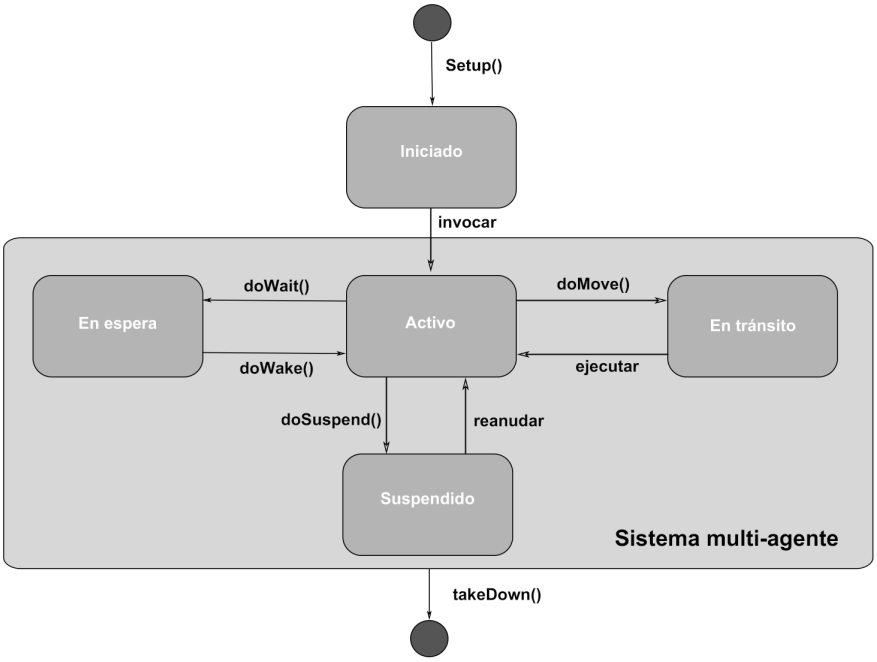


Figura 2.9: Las cajas negras representan los estados en los que se puede encontrar un agente, y las transiciones están etiquetadas con los métodos que permiten pasar de un estado a otro.

Es tarea del programador hacer un uso adecuado de estos métodos para cambiar el estado de los agentes según convenga según objetivos. El trabajo, funcionalidades y/o objetivos que tiene que realizar cada uno de los agentes, se encuentra dentro de sus comportamientos (o *behaviours*). Un comportamiento representa una tarea o servicio que un agente puede llevar a cabo, implementada como un objeto que extiende de la clase *jade.core.behaviours.Behaviour*. A fin de que un agente ejecute una tarea implementada en un comportamiento es suficiente con que el agente añada su comportamiento a su pila de comportamientos pendientes. Cada agente tiene asociado un planificador o *scheduler* de comportamientos que gestiona la entrada y salida de los mismos mediante una política round-robin sobre la cola FIFO donde se almacenan. Un comportamiento puede bloquearse mediante el método *block* y es entonces cuando se almacena en una segunda cola de la cual no sale hasta recibir la acción de desbloquearse. El siguiente esquema refleja el flujo de control de un agente sobre sus comportamientos.

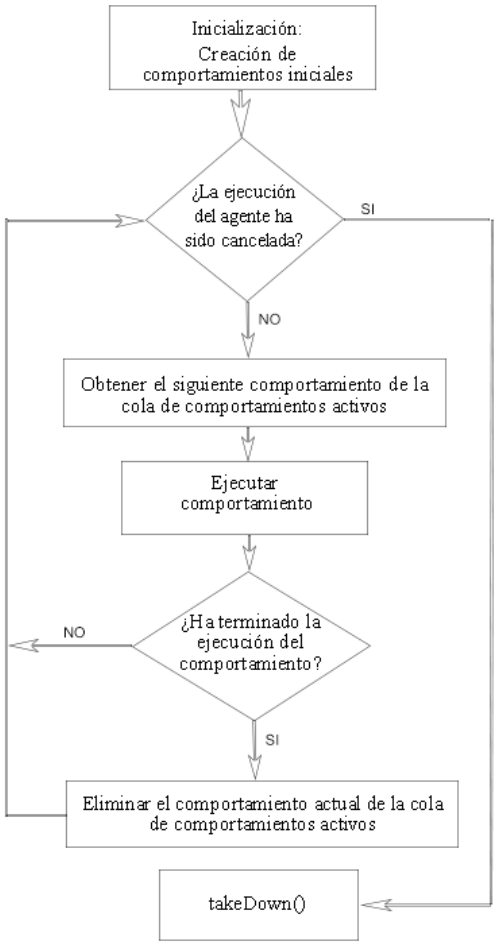


Figura 2.10: Flujo de trabajo de la gestión de los comportamientos de un agente.

Como refleja el esquema de la figura 2.10, en la fase de inicialización se lanzan una serie de comportamientos iniciales. Tras esto se comprueba si el agente sigue vivo, en cuyo caso se selecciona el primer comportamiento a ejecutar dentro de la cola de comportamientos pendientes y se ejecuta. En la siguiente fase se consulta si dicho comportamiento ha terminado. Es posible que no lo haya hecho ya que puede ocurrir que un comportamiento lance otros comportamientos o que el mismo comportamiento se deba ejecutar varias veces. Una vez que termina definitivamente su ejecución, se elimina el comportamiento de la cola de comportamientos activos y se vuelve a consultar si el agente continúa activo dentro del sistema. En el momento en el que el agente ya no se encuentre activo, pasará a eliminarse de la plataforma.

Un agente puede planificar la ejecución de varios comportamientos para que se ejecuten concurrentemente. Sin embargo, conviene recalcar que la planificación de comportamientos en un agente no es preventiva, sino cooperativa. Esto significa que cuando un comportamiento es planificado para ejecutarse, su método *action()* es ejecutado exclusivamente hasta que termina. Por lo tanto, no existe una concurrencia explícita como tal sino que es el programador quien puede decidir cuándo un agente cambia de la ejecución de un comportamiento a la ejecución de otro, pudiendo simular cierta concurrencia.

Existen distintos tipos de comportamientos que son accesibles desde el paquete *jade.core.behaviours* de JADE:

- Comportamientos simples: este tipo de comportamientos suelen representar tareas simples o atómicas.
  - OneShotBehaviour: Cuando el comportamiento se debe ejecutar sólo una vez de forma ininterrumpida y terminar.
  - CyclicBehaviour: Cuando el comportamiento debe ejecutarse una serie de veces. Este comportamiento se mantiene activo y consume recursos de CPU durante todo el tiempo que permanezca activo.
- Comportamientos compuestos: estos comportamientos se modelan a partir de la composición de otros subcomportamientos que pueden ejecutarse siguiendo diferentes políticas de planificación. Estas políticas vienen determinadas por la subclase de comportamiento escogida, la cual puede ser:
  - SequentialBehaviour: Esta subclase ejecuta los subcomportamientos de manera secuencial y una vez que todos han terminado termina su ejecución.
  - ParallelBehaviour: Permite que los subcomportamientos se ejecuten de manera concurrente y termina cuando se cumple determinada condición sobre la finalización de los subcomportamientos.
  - FSMBehaviour: Este comportamiento permite definir una máquina de estados finita mediante subcomportamientos. Cada subcomportamiento representa un estado de la máquina y las transiciones van ocurriendo según se producen las salidas de dichos estados.
- Comportamientos temporales: son comportamientos para ejecutar operaciones en determinados instantes de tiempo.
  - TicketBehaviour: Permite definir un comportamiento cíclico que ejecutará periódicamente una tarea. El período lo especifica el programador.

- WakerBehaviour: Este comportamiento implementa un comportamiento del tipo *OneShotBehaviour* que se ejecuta una vez que haya transcurrido un tiempo especificado.

2.2.2 Herramientas para programadores en JADE

Dentro de las funcionalidades de JADE, cabe destacar las herramientas para programadores que incorpora de manera visual mediante su interfaz gráfica generada por el agente RMA (Remote Agent Management) mostrada en la figura 2.11. Desde ella es posible realizar todo tipo de operaciones básicas sobre los agentes, explicadas en [34]. También existen operaciones más especializadas como por ejemplo, lanzar la GUI del agente DF (figura 2.12) mencionado en la sección anterior. Mediante el uso de esta interfaz, el usuario puede trabajar con las descripciones de los agentes y su registro, modificando o buscando las descripciones de ciertos agentes de interés o también es posible agrupar visualmente el DF con otros DF's creando jerarquías o redes de dominios primarios y secundarios de páginas amarillas.

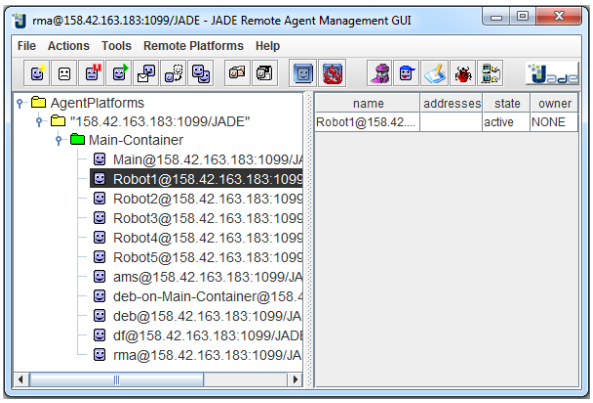


Figura 2.11: Interfaz gráfica principal de la plataforma JADE ofrecida por el agente rma.

Otra herramienta muy útil es el agente *Introspector*. Este agente, cuya interfaz se muestra en la figura 2.13 permite supervisar y controlar el ciclo de vida de cualquier agente en ejecución, así como los comportamientos que ejecuta y la cola de sus mensajes enviados y recibidos. Además es posible cambiar el estado de un agente mediante un clic y añadir o quitar comportamientos, descartar o visualizar mensajes, etc., de una manera gráfica e intuitiva.

JADE también ofrece desde su interfaz gráfica la posibilidad de utilizar la herramienta *DummyAgent*, la cual extiende de la clase *jade.tools.DummyAgent.DummyAgent* incluida dentro de los fuentes de JADE. Esta herramienta se puede lanzar desde la

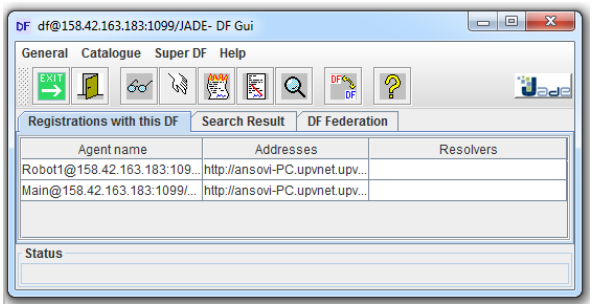


Figura 2.12: Interfaz gráfica del agente DF de la plataforma JADE.

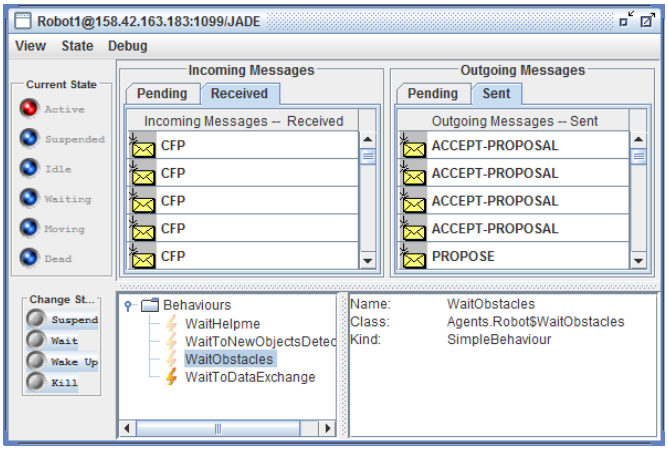


Figura 2.13: Interfaz gráfica del agente *Introspector* de la plataforma JADE.

propia interfaz de JADE y facilita por ejemplo, la validación de la programación del agente antes de integrarlo dentro de un MAS específico. Es posible comprobar la respuesta del agente cuando recibe ciertos mensajes, así como sus cambios de ciclo de vida o el estado de sus comportamientos ya que esta IGU permite enviar y recibir mensajes. Además, es posible visualizar el contenido de los mensajes, lo cual resulta de gran utilidad siempre que el contenido del mensaje sea legible, es decir sea atómico o contenga un string. La imagen 2.14 muestra dicha interfaz y sus parámetros de configuración, como el nombre del receptor, el lenguaje, el protocolo utilizado en la conversación, etc. Además también es posible recibir mensajes de cualquier agente o cargar o guardar mensajes desde el disco duro, lo cual resulta muy útil para evitar operaciones repetitivas y poder guardar un registro del comportamiento de los protocolos de comunicación.

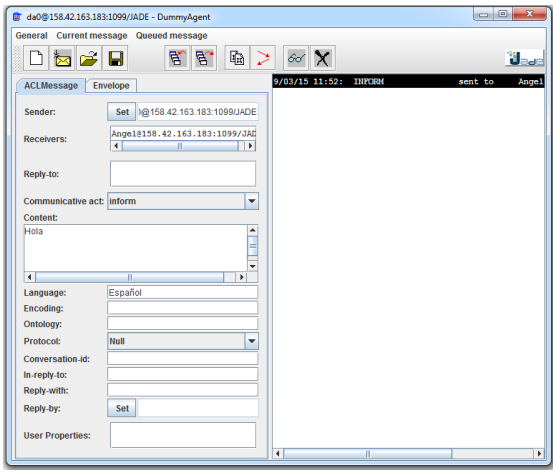


Figura 2.14: Interfaz gráfica del *DummyAgent* de la plataforma JADE.

Por último, una de las herramientas más utilizadas para la verificación de las comunicaciones entre agentes, es el *Sniffer Agent*. Este agente puede funcionar sobre cualquier instancia de la plataforma o contenedor y muestra visualmente las interacciones entre los distintos agentes de una manera individualizada y en tiempo real como muestra la figura 2.15. Es necesario seleccionar los agentes de interés para mostrarlos dentro del monitor. El resto de agentes no seleccionados se representan con la etiqueta común *Other* y no se muestra información detallada sobre su interacción.

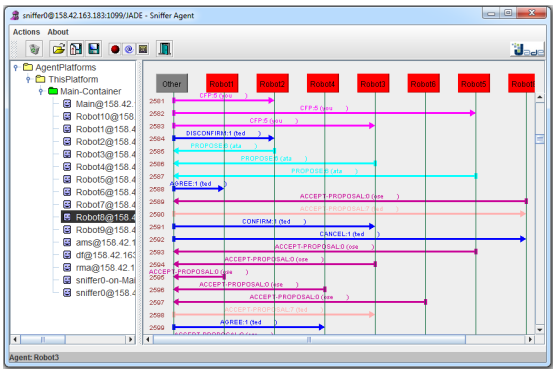


Figura 2.15: Interfaz gráfica del agente *sniffer* donde se visualizan las distintas interacciones entre agentes/robots.

En el caso de la figura 2.15 se puede observar que cada mensaje está etiquetado con un identificador que se incrementa de manera secuencial en el tiempo. Los distintos protocolos se colorean de manera distinta y los mensajes se representan con flechas direccionales, lo cual permite identificar rápida y visualmente el emisor, el receptor y la extensión de la conversación.

A pesar de la disponibilidad de estas herramientas, para el desarrollo de esta tesis ha sido necesaria la elaboración de una herramienta mejorada de depuración y verificación del sistema la cual se expone en la siguiente sección como la primera aportación directa de la presente tesis al campo de la investigación en sistemas multi-agente y en concreto a la utilización de la plataforma JADE.

2.2.3 Aportaciones a la plataforma JADE

Sin duda, una de las dificultades inherentes a los sistemas distribuidos es la complejidad de su supervisión. Supervisar suele requerir centralizar y ésto es lo que precisamente intentan evitar dichas arquitecturas. La monitorización y depuración de procesos en sistemas distribuidos puede resultar una tarea muy compleja dada la alta concurrencia del sistema, donde múltiples agentes se ejecutan con distintas funcionalidades y distintos procesos. Si se tiene en cuenta que dichos procedimientos pueden depender del satisfactorio resultado de una conversación entre agentes, surge la necesidad de poder supervisar también dichas conversaciones. Además, en la mayoría de los casos no sólo interesa conocer el momento o la secuencia de la interacción entre los agentes, si no que muchas veces el significado semántico de los mensajes es lo que deriva el sistema de un estado a otro, lo cual aumenta sustancialmente el grado de complejidad de la supervisión.

Como se ha descrito en la sección anterior, JADE ofrece algunas herramientas para la depuración y monitorización de las comunicaciones del sistema como son el agente *Introspector*, enfocado a la visualización de los buzones de entrada y salida de cada agente, y agente *sniffer* enfocado a la observación de la interacción comunicativa entre varios agentes.

En la interfaz del agente *Introspector* se muestra una lista con todos los agentes que están registrados en el sistema. Para acceder a la información de sus buzones de entrada y salida, es necesario seleccionar el/los agente/s de interés uno por uno y seleccionar la opción de *Debug On* a través de la interfaz como muestra la figura 2.16. Cada vez que se activa la opción de depurar un agente, se abre una nueva ventana dentro de la IGU del agente *Introspector* con la información solicitada. Cuando se activa la depuración de múltiples agentes, las ventanas comienzan a sobreponerse unas con otras y resulta muy complicada la visualización conjunta de todas al mismo tiempo. Un ejemplo de ello se muestra en la figura 2.17 donde se ha activado la depuración de 5 agentes y por mucho que el programador o el

usuario intente ajustar las ventanas, no es posible visualizar la información de los 5 a la vez de una manera cómoda.

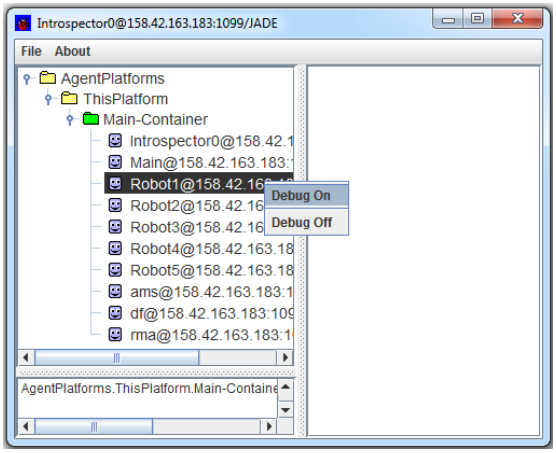


Figura 2.16: Activar la depuración de un agente desde la interfaz del agente *Introspector*.

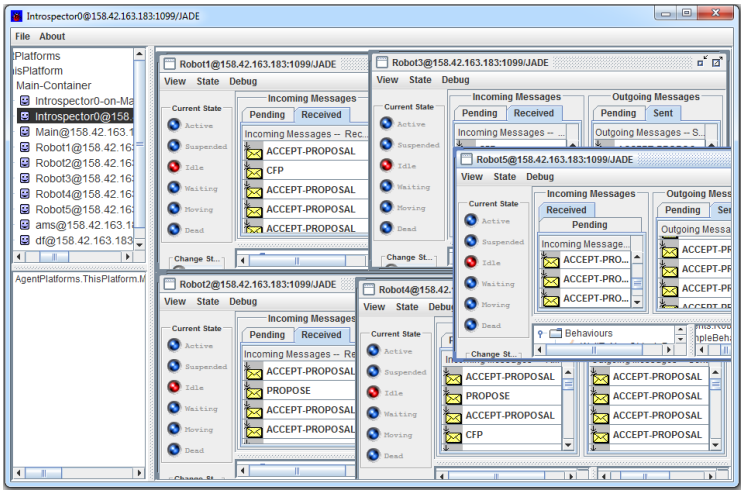


Figura 2.17: Visualización de la depuración de cinco agentes mediante el agente *Introspector*

Existen, entre otras, dos importantes limitaciones que comparten estas dos herramientas ofrecidas por JADE:

- *Necesidad del registro previo en el sistema para su depuración.* En ningún momento es posible poner en modo de depuración un agente que todavía no se encuentra dentro del sistema porque no existe la manera de seleccionarlo desde la interfaz. Esto supone un problema en la mayoría de casos donde se requiere una depuración inmediata a la creación del agente ya que en el momento del inicio de su ejecución, el agente ya puede comenzar a comunicarse con el resto y dicha comunicación no será posible registrarla desde el propio agente. Para poder monitorizar las comunicaciones de un agente inmediatas a su creación mediante estas herramientas, haría falta detenerlo temporalmente vía software o mediante la espera de un determinado evento, para dar tiempo al programador a activar la opción de depuración desde la interfaz. Ésto no resulta lo más apropiado y menos en sistemas complejos con múltiples agentes o sistemas donde la creación y destrucción de agentes se produce frecuentemente.

- *No reconocimiento de agentes que en algún momento ya formaron parte del sistema.* Estas herramientas tampoco son capaces de reconocer un agente el cual ya había sido registrado en el sistema anteriormente durante la misma ejecución de la plataforma, a pesar de que mantenga el mismo nombre. Es decir, si a un agente registrado dentro del sistema, se le activa la opción de depuración desde cualquiera de estas dos herramientas y posteriormente el agente termina su ejecución pero más tarde vuelve a registrarse en el mismo sistema y bajo el mismo nombre, estas herramientas no son capaces de reconocerlo y no mostrarán información de depuración relativa a su nueva ejecución. La depuración finaliza en el momento en el que el agente termina y no se vuelve a reanudar. Para tal caso es necesario volver a activar la depuración manualmente, buscando de nuevo el agente dentro de la lista de agentes activos de la interfaz. Esto supone un problema serio ya que en muchas ocasiones, sobre todo en casos de negociación o toma de decisiones conjunta, los agentes involucrados crean nuevos agentes temporales que les representan y se encargan de comunicarse con el resto de agentes para llegar a algún tipo de conclusión que finalmente le comunican al agente que representan. Una vez terminan su cometido, estos agentes representantes acaban su ejecución con el fin de liberar recursos dentro del sistema. Estas situaciones se pueden repetir en multitud de ocasiones dentro de un mismo experimento por lo que a la hora de depurar las comunicaciones de esos agentes representantes, las herramientas que ofrece JADE no son suficientes.

Con el objetivo de suplir las carencias mencionadas anteriormente, se ha desarrollado la interfaz de un nuevo agente denominado *debugger* que ofrece nuevas funcionalidades y además intenta concentrar la información más relevante de los agentes *Sniffer* y *Introspector* en una única ventana con una organización más orientada a la depuración de múltiples agentes a la vez.

El agente *debugger* se ha programado de tal manera que es compatible con cualquier sistema desarrollado mediante la plataforma JADE. A nivel de software, el agente se ha compilado dentro de una librería estándar de JAVA en formato *.jar*, de modo que es posible integrar dicha librería en cualquier proyecto desarrollado en lenguaje JAVA. No se requiere ningún tipo de integración o configuración adicional para su uso. Desde la misma interfaz que ofrece JADE con el agente RMA es posible lanzar el agente *debugger* seleccionando la librería desarrollada.

Cuando dicho agente inicia su ejecución, se suscribe al agente AMS para obtener información en tiempo real de los agentes activos en el sistema y lanza la interfaz mostrada en la figura 2.18. Conforme el usuario o programador activa la depuración de los agentes, el *debugger* va suscribiéndose a cada uno para obtener una copia del mensaje que envían o reciben de manera pasiva, sin interferir en las conversaciones del sistema. Es decir, cuando un agente entra en modo de depuración, su tráfico de mensajes se duplica para hacerlo llegar tanto al agente receptor original, como al agente *debugger*. De esta forma, todos los mensajes de interés se reciben en el *debugger* de manera centralizada y es posible monitorizarlos de un modo más sencillo.

Tal y como se muestra en la figura 2.18, la interfaz de este agente *debugger*, se ha dividido en tres ventanas principales. En la ventana de la izquierda se listan los contenedores de la plataforma y los agentes registrados dentro de cada uno. Cada agente tiene un *checkbox* para activar o desactivar su depuración en tiempo real. Además, seleccionando un agente con el botón derecho del ratón se despliega un menú con las opciones de activar, terminar o suspender el estado del agente. Este menú ofrece también la posibilidad de obtener el estado actual de cualquier agente para que se muestre en una ventana nueva y también de personalizar el color con el que se muestran los agentes al activar su depuración.

La parte derecha de la interfaz muestra dos ventanas divididas de forma horizontal. En la ventana superior se muestra de manera individual el estado de cada agente que se desea depurar y los comportamientos que tiene almacenados en su pila de comportamientos activos. Cada vez que se ejecuta un comportamiento, su nombre se sombrea para que el usuario pueda identificarlo en tiempo real. También se ha incluido la funcionalidad de mostrar los buzones de entrada y salida de mensajes que ofrece el agente *Introspector* de JADE dentro del botón "*Show Messages*". A diferencia del *Introspector*, dicha información se lanza en una nueva ventana independiente que el usuario puede situar en su monitor de la manera que considere más adecuada.

La ventana inferior derecha muestra los agentes que tienen activado el modo de depuración a modo de círculos de colores con su nombre encima. Cada agente se representa por duplicado, una vez en la parte superior como emisor y otra en la inferior como receptor. Cuando se ejecuta el *debugger* y se ponen algunos agentes en modo de depuración, el número de mensajes que dichos agentes envían o reciben se

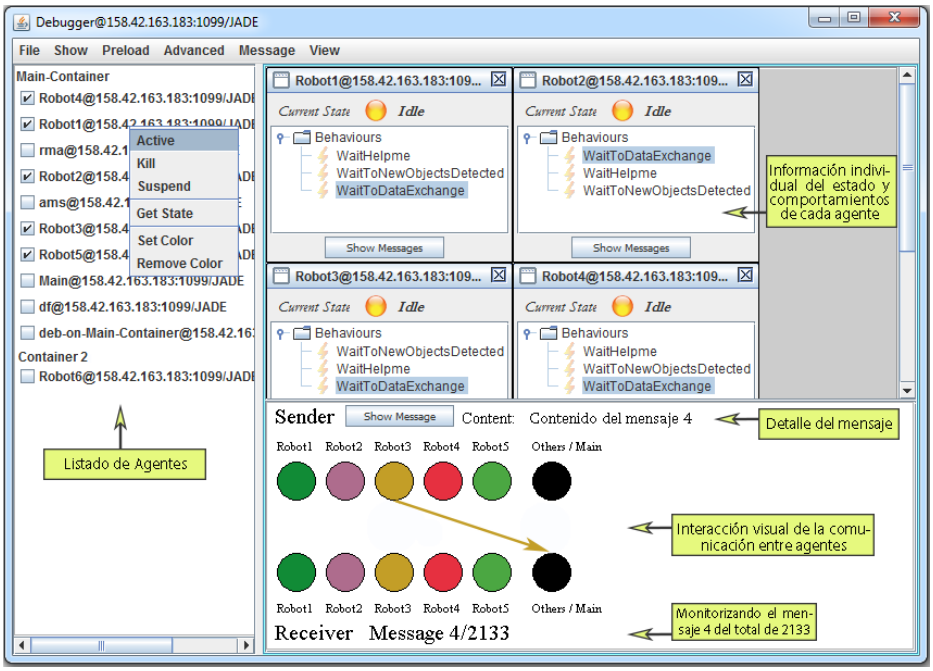


Figura 2.18: Interfaz desarrollada para la depuración y monitorización de las comunicaciones entre agentes en JADE

van almacenando en la parte inferior de la ventana. Llegado el momento de querer analizar dichos mensajes con detalle, es posible reproducir cada mensaje uno a uno. De esta manera, por cada mensaje depurado se dibuja una flecha que empieza en el agente emisor y termina en el agente receptor. El contenido del mensaje se muestra en la parte superior de la ventana y es posible acceder a sus detalles pulsando el botón "Show message". Se han añadido varias funcionalidades como el hecho de poder configurar la velocidad de reproducción de las comunicaciones que se están visualizando o la posibilidad de poder reproducirlas paso a paso, pudiendo volver atrás o adelante en cualquier momento. Además de los agentes en depuración, en esta ventana siempre se muestra un círculo negro con la etiqueta "Others"que representa el resto de agentes que no están incluidos en la representación. De modo que si un agente se comunica con otro agente que no se encuentra en modo de depuración, una flecha señalará el círculo negro para indicar que el mensaje va dirigido a otro agente que actualmente no se muestra porque no se está depurando. En la figura 2.18 se puede observar este hecho donde la flecha dibujada indica que el agente con nombre "Robot3"le ha enviado un mensaje al agente "Main", el cual como no se encuentra en modo de depuración, está incluido en la etiqueta "Others". Ya que puede resultar interesante almacenar toda esta información, se ha incluido

en la interfaz la posibilidad de imprimir en cualquier momento un registro o log completo de toda la depuración tanto en forma de fichero de texto como mediante su visualización por consola.

Otra de las funcionalidades implementadas más importantes es la posibilidad de poder precargar la depuración de un agente que todavía no existe en el sistema. Para ello la interfaz solicita al usuario el nombre del agente o los agentes de interés aceptando además caracteres especiales dentro del nombre como es el caso de los asteriscos. Si por ejemplo se introduce como nombre la cadena "*robot**", el agente *debugger* tendrá en cuenta a partir de ese momento que debe poner en modo de depuración cualquier agente que su nombre empiece por la cadena "*robot*". De este modo esta herramienta es capaz de depurar un agente desde el primer momento que entra en ejecución lo cual suple la carencia comentada anteriormente de los agentes *Introspector* y *Sniffer*.

Además, se ha implementado la capacidad de reconocer agentes que ya estaban en el sistema pero terminaron su ejecución por algún motivo y volvieron más tarde a registrarse bajo el mismo nombre dentro del sistema. En este caso el registro de los mensajes se muestra y se guarda como si dicho agente no hubiera terminado nunca su ejecución.

La herramienta de depuración y monitorización *debugger*, descrita en el presente capítulo ha sido de gran ayuda y ha facilitado la comprobación de todos los experimentos desarrollados durante la presente tesis. Siguiendo la filosofía de la plataforma JADE, dicha herramienta se ha alojado en un servidor web y se distribuye de manera libre y gratuita bajo una licencia *Creative Commons BY-NC-SA* para cualquier persona que le pueda interesar.

2.3 Frameworks de Software Orientados a la Robótica

Como es bien sabido los RSFs intentan ofrecer una abstracción entre el hardware de un robot y su programación software. En la presente tesis se ha trabajado con distintos RSFs según las distintas plataformas utilizadas para cada experimento. En algunos casos, para cada robot utilizado existen múltiples RSFs disponibles, algunos ofrecidos por el fabricante y otros desarrollados por terceros con la intención de facilitar su programación u ofrecer nuevas funcionalidades u opciones de programación.

A continuación se detallan los dos RSFs más populares sobre los que se ha trabajado: el firmware *LeJOS* para el robot LEGO Mindstorms y el framework *ROS* soportado por diversas plataformas utilizadas como demostradores en la presente tesis como el robot Summit XL o Robotino®.

2.3.1 LeJOS

LeJOS (LEGO Java Operating System) es un firmware para LEGO Mindstorms basado íntegramente en el lenguaje de programación Java. Como es bien conocido, Java provee de una máquina virtual que permite ejecutar código compilado Java sea cual sea la plataforma que exista por debajo. Es por eso que este RSF no requiere un sistema operativo concreto para funcionar como por contra ocurre con otros RSFs.

Además, leJOS es un software totalmente libre por lo que se puede descargar desde su página web en su versión más completa `/www.lejos.org`, ofreciendo incluso el código fuente de todas las librerías que implementa. Esto ha facilitado que haya disponible una gran cantidad de proyectos de libre distribución y código abierto por parte de la comunidad desarrolladora.

Junto con las librerías que componen el software de leJOS, se adjuntan herramientas de compilación, depuración de programas, manejo de ficheros dentro del robot, gestión de la conectividad inalámbrica, monitorización en tiempo real, etc. Entre dichas herramientas se encuentra el programa *nxjflash.exe*, el cual se utiliza para llevar a cabo el cambio del firmware del robot, ya que es necesario sustituirlo para poder utilizar leJOS.

Las opciones para la programación en leJOS, son las mismas que para cualquier programa en Java. Los dos IDEs de distribución libre más utilizados para Java son Eclipse y NetBeans. No obstante, la comunidad de leJOS ha desarrollado un plugin para Eclipse el cual proporciona algunas funcionalidades directas sobre el robot, como cargar los programas directamente, configurar automáticamente proyectos para el Lego o cambiar el firmware del robot pulsando tan sólo en un icono, entre otras.

Por otro lado, al contrario de como ocurre en otros RSFs, las librerías de leJOS dan soporte tanto al hardware original de Lego (sensores y motores) como al desarrollado por otras compañías o usuarios. Durante el desarrollo de esta tesis por ejemplo, se ha integrado de manera física al LEGO y a modo de librería externa a leJOS, unos sensores de distancia mediante infrarrojos construidos y desarrollados desde cero, lo cual no habría sido posible realizar con otros RSFs que no permiten un acceso a tan bajo nivel de programación del hardware.

Con este RSF, los programadores de los robots pueden trabajar con abstracciones de alto nivel aprovechándose de su estructura orientada a objetos de Java, lo que evita tener que enfrentarse con detalles de bajo nivel como por ejemplo las direcciones hexadecimales de los componentes hardware. De esta forma leJOS no sólo incluye la implementación de controladores tipo PID y filtros de Kalman, sino que se tienen librerías con funciones más abstractas como la navegación, la cartografía, la programación basada en comportamientos, etc.

Para la nueva versión de lego Mindstorms EV3, leJOS ha conseguido situarse como la alternativa de desarrollo más completa respecto al firmware original ofrecido por LEGO, ofreciendo hoy en día la total funcionalidad de sus librerías. No obstante, no cabe duda de que pronto surgirán otros RSFs alternativos como ocurrió en su día con su versión anterior NXT.

2.3.2 ROS

Entre los middlewares de control de robots o RSFs más extendidos y populares hoy en día, ROS (Robotic Operating System) destaca por estar basado en una arquitectura distribuida, completamente modular, y disponer de una comunidad de desarrolladores muy activa. Comparte algunas características de otros RSF como Player, YARP, Orocos, CARMEN, Orca, MOOS o Microsoft Robotics Studio. No obstante, ROS es un software de código totalmente abierto que ofrece a cualquier desarrollador la posibilidad de compartir sus aportaciones con la comunidad. De ese modo, los investigadores no están reinventando la rueda constantemente, si no que pueden partir sus investigaciones desde algoritmos que ya han sido probados y verificados.

ROS soporta la programación en C++, Python, Octave, LISP e incluso JAVA. Está desarrollado bajo licencia Creative Commons Attribution 3.0, lo que permite su libre uso, distribución y modificación de todo el código con la condición de que se especifique siempre a los desarrolladores originales.

A continuación se muestra una breve descripción de los principales componentes de ROS:

- **Nodo ROS Master:** Se trata del proceso principal de la arquitectura. Es quien permite que los nodos se encuentren entre sí e intercambien mensajes. Se podría decir que es la pizarra donde los nodos se registran y a la vez buscan lo que les hace falta.
- **Nodos:** Un nodo es el mínimo proceso de cómputo de la arquitectura. Los nodos constituyen los principales componentes del Framework de ROS. Dado su diseño modular, el sistema de control de un robot suele estar compuesto por diversos nodos, donde cada uno lleva a cabo una función concreta. Estos nodos se pueden escribir tanto en C++ como en Python utilizando las librerías ofrecidas por ROS.
- **Topics:** Los topics son el principal medio de comunicación entre nodos de ROS. Unos nodos pueden publicar mensajes mediante topics a una determinada frecuencia y otros nodos pueden suscribirse a dichos topics para recibirlos. De esa manera se establece una comunicación asíncrona de información entre nodos.

- **Servicios:** Los servicios se encargan de las interacciones que no pueden resolverse mediante topics. También son publicados por nodos, sin embargo son síncronos y responden a una comunicación mediante eventos, es decir, de llamada y respuesta.

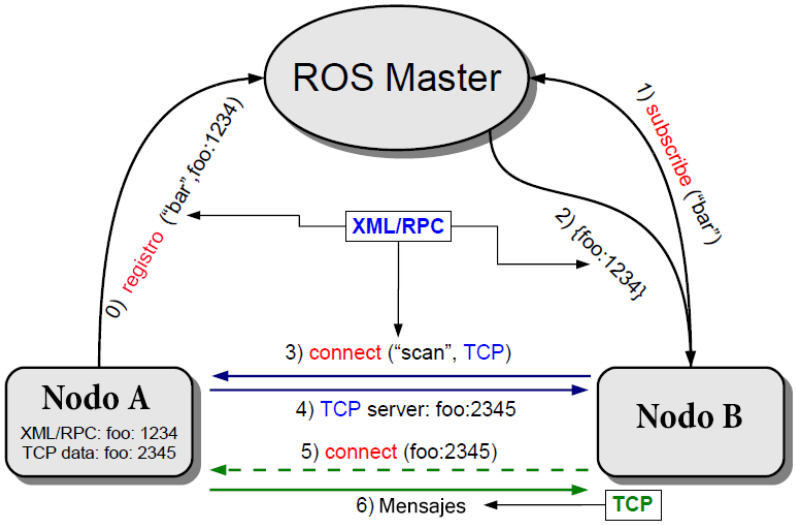


Figura 2.19: Funcionamiento del protocolo de comunicación de ROS

La figura 2.19 muestra un ejemplo de la estructura que tiene una arquitectura en ROS formada por un ROS master y dos nodos ($NodoA$ y $NodoB$). Cuando el $NodoA$ se inicia, se registra en el ROS Master y éste anota el nombre del topic que publica (denominado *bar* en el ejemplo), su ROS_HOSTNAME (foo) que lo identifica como si fuera una IP única y el puerto del nodo publicador (*1234* en este caso). En el siguiente paso (1 en la figura), el $NodoB$ se inicia y se registra en el ROS Master, solicitando además una suscripción a un topic de nombre *bar* y el ROS Master le envía el ROS_HOSTNAME y el puerto del nodo que lo publica (paso 2). De esta manera el $NodoB$ solicita una conexión de tipo TCP al $NodoA$, quien responde con el puerto de su servidor TCP (*2345*). Es entonces cuando el $NodoB$ envía una solicitud de conexión al servidor TCP y el $NodoA$ la acepta, creándose de ese modo el canal de comunicación directo entre los dos nodos. Como se puede observar el ROS Master gestiona tan sólo el vínculo de la comunicación pero después ya no interviene a menos que se realice una consulta distinta. Es por este motivo por el que una vez abierto el canal, si el ROS Master se cae, la comunicación entre los dos nodos continúa funcionando.

2.3.3 V-REP

Existen distintos simuladores orientados a los MAS tales como MASON [107], SeSAm [89], breve [87] o FLAME [147] entre otros. Sin embargo V-REP ([68]) surge en 2010 como uno de los primeros simuladores orientados al área de la robótica y con una arquitectura de control totalmente distribuida, lo cual resulta idóneo dentro del marco en el que se desarrolla esta tesis.

V-REP es un simulador 3D desarrollado por la empresa suiza Coppelia Robotics (`/www.coppeliarobotisc.com`) donde cada objeto o modelo robótico puede ser controlado de manera independiente al resto y con metodologías y comunicaciones distintas. El esquema mostrado en la figura 2.20 ha sido extraído de la documentación de V-REP y ofrece una visión de los diversos mecanismos de comunicación y mensajería que integra V-REP. De los distintos modos de programación caben destacar tres por haber sido los más utilizados en este trabajo:

- *Embedded Script*. Este método consiste en la escritura de scripts bajo el lenguaje de programación imperativo Lua. Es posible personalizar la simulación a través de secuencias de comandos basados en la API proporcionada por V-REP, la cual incluye tanto el control de robots como métodos de realimentación de la posición del robot dentro del escenario de simulación y otros.
- *Remote API*. Este método permite a una aplicación externa conectarse a V-REP utilizando unos comandos específicos de la API de V-REP. Este ha sido el método utilizado para la integración de V-REP con JADE.
- *ROS node*. Como su nombre indica, esta metodología ofrece la posibilidad de poder controlar la simulación de cualquier robot a través de un nodo de ROS el cual se comunica directamente a través de un topic al programa V-REP. Del mismo modo que ocurre con el método anterior, el control de cada robot es individualizado y se puede realizar mediante topics distintos.

V-REP no sólo funciona a modo de simulador sino que incluye también tres motores físicos diferentes para el tratamiento de detección de colisiones entre objetos, simulación de la dinámica de cuerpos rígidos y emulación de propiedades físicas. Estas características dotan al software de la capacidad de emular ejecuciones de modo que es posible obtener una realimentación de dicha emulación. Por ejemplo, es posible utilizar un robot real y su correspondiente modelo dentro de V-REP de manera que, en tiempo real los dos respondan al mismo sistema de control. O, por ejemplo, que bajo la misma situación, el robot real responda según las medidas de un láser incorporado tan sólo en su modelo de V-REP.

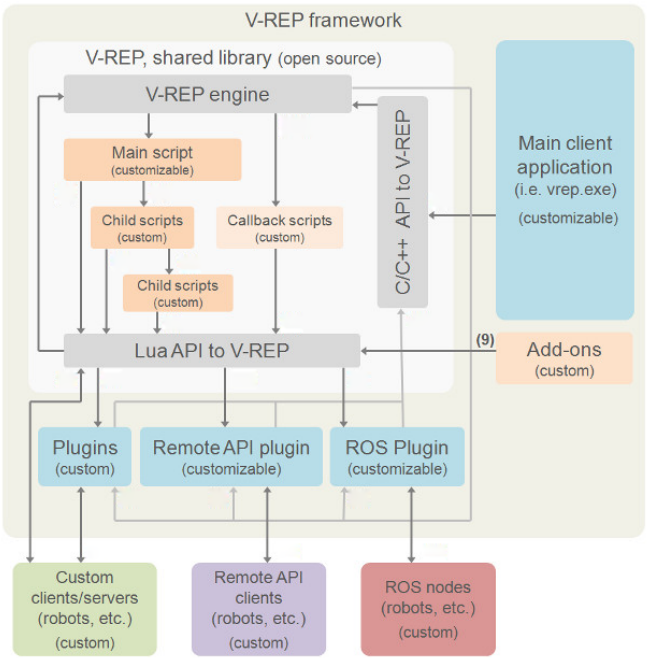


Figura 2.20: Diagrama de las comunicaciones de V-REP y de las posibilidades de interconexión con otros programas externos

2.4 Aportación a la Integración de MAS con RSF

Aún hoy en día, cuando se trata de buscar demostradores para aplicaciones sobre inteligencia artificial o sobre teoría de agentes, no existen plataformas hardware adaptadas a su aplicación directa, por lo que normalmente siempre se trabaja con simulaciones o aplicaciones que difícilmente llegan a reproducirse en algún tipo de hardware o robot físico real. Además, cuando se trabaja con este tipo de hardware físico, siempre surgen ciertas problemáticas inherentes a este tipo de sistemas, por lo que es también de interés poder separar e independizar el desarrollo software basado en los RSF de las plataformas físicas, del desarrollo software basado en las estrategias de los MAS.

La presente tesis aporta el desarrollo de un middleware de control y comunicación capaz de brindar la posibilidad de aplicar algoritmos basados en la teoría de agentes a sistemas robóticos reales. Además, aporta una abstracción completa que permite aislar e independizar la programación mediante el RSF de las funcionalidades específicas de un robot individual, de las funcionalidades del agente que lo representa dentro del MAS. Esto permite que puedan existir dos perfiles de programador en el

sistema de manera que cada uno esté especializado dentro de su campo y no deba conocer necesariamente ambos. El programador o administrador del MAS se puede centrar en la problemática de otorgar inteligencia, cooperatividad y coordinación al sistema sin preocuparse por el tipo de programación del robot al que representa cada agente, mientras que la programación del RSF específico de cada robot queda a un nivel más bajo dentro de la arquitectura y puede ser llevada a cabo por otro programador especializado en dicho RSF. La figura 2.21 muestra el esquema que sintetiza esta idea donde se exponen cuatro robots utilizados para el desarrollo de los demostradores de esta tesis, donde cada uno posee una configuración distinta, unas funcionalidades distintas y un RSF distinto.

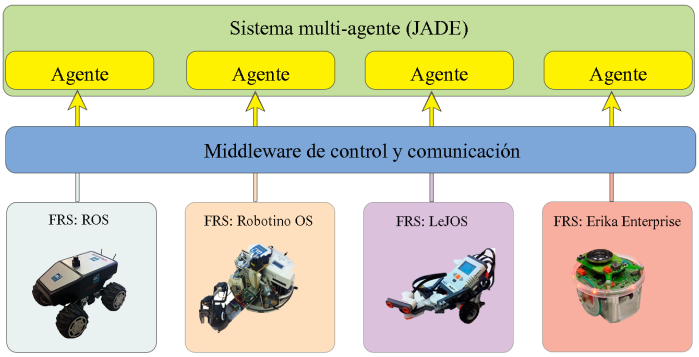


Figura 2.21: Esquema conceptual sobre la integración mediante el middleware de control y comunicación desarrollado, el cual consigue que cada robot forme parte del MAS de una manera abstraída totalmente de su RSF.

A pesar de que en la sección de demostradores de esta tesis se describen más detalladamente las plataformas utilizadas y sus características físicas principales, si se observa de izquierda a derecha la figura 2.21, el primer robot que se muestra es el Summit XL desarrollado por la empresa Robotnik y cuyo RSF incorporado de serie es *ROS*, descrito brevemente en la sección anterior. El siguiente robot es Robotino® de la empresa Festo el cual por defecto incorpora el *Robotino OS* pero también soporta otros RSFs entre los que se incluye *ROS* o *Microsoft Robotics Developer Studio (MRDS)*. LEGO Mindstorms es el robot que se muestra a continuación el cual para el desarrollo de esta tesis funciona bajo el firmware *LeJOS* el cual también se ha descrito con anterioridad; y por último, se muestra el robot e-puck desarrollado por el Instituto Federal Suizo de Tecnología de Lausanne (EPFL) cuyo RSF original se ha reemplazado por el sistema operativo Erika Enterprise de tipo OSEK/VDX (`/http://erika.tuxfamily.org/drupal/`).

Todos estos RSFs han sido integrados dentro del MAS JADE a través del middleware desarrollado. A continuación se describe la justificación y el desarrollo de dicha integración sobre los RSF de leJOS y ROS.

2.4.1 Middleware JADE-LeJOS

Durante los últimos años, la plataforma LEGO Mindstorms NXT entre otras, ha sido una herramienta muy utilizada tanto en investigación como en la educación y aprendizaje de la mecatrónica y robótica, tal y conforme se recoge en [184] o [183]. Uno de los motivos principales que justifican el interés de la integración de su RSF (en este caso, leJOS) con un MAS como JADE, es aprovechar la popularidad de dicha plataforma para poder hacer uso de la misma en aplicaciones sobre inteligencia artificial.

El middleware JADE-Lejos (en adelante *J-L*) desarrollado en la presente tesis ofrece la posibilidad de utilizar las plataformas LEGO como demostradores de nuevas estrategias aplicadas a la teoría de agentes y orientadas a su aplicación en el área de la robótica.

Algunos RSFs como leJOS, ofrecen la posibilidad de intercomunicar varios robots entre sí, sin embargo, el número de robots conectados suele estar limitado y existen escasas herramientas para gestionar dichas comunicaciones. El middleware *J-L* ofrece la ventaja de no tener límite de conexiones en el número de robots interconectados y además poder hacer uso de los protocolos de conversación que otorga JADE para la gestión de las comunicaciones.

En cuanto a la integración software del middleware sobre la versión NXT del robot LEGO Mindstorms, no resulta fácil incorporar nuevas librerías personalizadas dentro del software del ladrillo ya que la memoria es muy limitada y la única manera factible de hacerlo es únicamente mediante la modificación directa del firmware completo. Aún así, para los primeros experimentos de la presente tesis con los robots LEGO Mindstorms NXT se han integrado dichos robots con JADE mediante el middleware *J-L*, el cual hace uso de un computador externo donde se ejecuta el contenedor donde reside el agente robótico que representa el NXT dentro del MAS. El middleware contiene un protocolo de comunicaciones definido tanto en la parte del robot como en la del PC, de modo que el robot puede actuar sobre la plataforma JADE mediante su comunicación con el PC. Un problema añadido es que estos robots tan sólo implementan el protocolo inalámbrico Bluetooth en su versión 2.0, lo que ofrece unos tiempos de ida y vuelta de aproximadamente unos 200 ms., por lo que a menudo existe un importante retardo en las comunicaciones que hay que tener en cuenta.

La aparición reciente de la nueva versión de la plataforma LEGO Mindstorms EV3 la cual incorpora la capacidad de poder cargar un ligero sistema operativo basado en Linux mediante una tarjeta microSD, conexión Bluetooth 4.0 y conectividad Wi-Fi de alta velocidad, ha abierto nuevas alternativas a su integración en los MAS. En este caso, ha sido posible incluir el middleware *J-L* completo más la plataforma jade completa dentro del propio software del robot, lo que le otorga

una independencia absoluta a diferencia de como ocurría con el NXT y ofrece unos retardos mínimos de comunicación entre agentes.

Esta opción no sólo ofrece el hecho de que cada robot pueda ejecutar internamente un contenedor dentro de la plataforma si no que él mismo puede ejecutar diversos agentes internos que se encarguen de tareas más simples como el movimiento de los motores, o la lectura de un sensor, como suele ocurrir en el caso del framework ROS.

Tal y conforme se verá más adelante del presente documento, gracias al desarrollo del middleware *J-L*, se han podido realizar demostraciones sobre algoritmos desarrollados en esta tesis implementados sobre más de seis robots LEGO Mindstorms NXT que interactúan, se comunican y colaboran entre sí dentro del MAS Jade.

2.4.2 Middleware JADE-ROS

ROS se ha convertido sin duda en uno de los RSFs más populares de la actualidad. Se ha implantado en más de 50 tipos de robots diferentes y soporta hoy en día más de 50 sensores distintos. Sin embargo, a día de hoy sigue orientado hacia la programación local distribuida de un robot individual y no ofrece un buen soporte ni una propuesta detallada para su utilización en casos de sistemas multi-robot.

Para instaurar una arquitectura basada en ROS, es necesario que exista un único nodo dentro del sistema que ejecute el *ROS MASTER* y cuya dirección debe ser conocida por el resto de nodos. El *ROS MASTER* es estrictamente necesario para la coordinación inicial entre nodos ya que contiene la información centralizada sobre el identificador y los topics y servicios que ofrece cada nodo. Como se explicó en la sección 2.3.2, una vez se establecen las comunicaciones entre nodos, si el proceso del *ROS MASTER* termina o muere, el sistema puede continuar funcionando pero no será posible descubrir nuevos nodos o modificar las suscripciones a los mismos, por lo que esta situación no suele ser deseable.

Teniendo en cuenta esta restricción, se plantean dos opciones a la hora de definir una arquitectura para la implantación de ROS en aplicaciones multi-robot:

- **Unimaster.** Dado que no puede existir más de un nodo *ROS MASTER* dentro del sistema, es posible plantear una arquitectura donde exista un robot o una estación de trabajo la cual sea la encargada de dicha ejecución, y el resto de robots o sistemas, conozcan su dirección y la utilicen para realizar sus consultas. La desventaja del uso de esta solución es que se crea una fuerte dependencia con la cobertura de la red y el correcto funcionamiento del canal de las comunicaciones. Además la coordinación a priori de la arquitectura estaría centralizada en el sistema que ejecutase el *ROS MASTER*, lo cual se contradice con la filosofía de los sistemas distribuidos.

- **Multimaster.** La comunidad de desarrollo de ROS, consciente de la necesidad de no restringir la ejecución de un solo nodo *ROS MASTER*, creó un grupo de interés en el desarrollo de una solución de alto nivel basada en la ejecución de varios nodos *ROS MASTER*. De este modo, cada robot puede ejecutar un máster individual y al mismo tiempo es posible establecer una comunicación entre masters. Hoy en día el sitio ROS ofrece tres posibles paquetes de soluciones:

  - *Multimaster*. Este paquete debe estar funcionando en cualquier máster con el que se desee interactuar. No ofrece ningún servicio para descubrir nuevos másters sino que requiere conocer las direcciones de los otros masters a los que se desea conectar. Es posible solicitar los topics y servicios de masters externos sin embargo la documentación sobre sus funcionalidades es muy escasa. Además, dicho paquete no se actualiza desde la versión *fuerte* de ROS, de mediados del 2012.

  - *App_manager*. Este paquete se desarrolló para la versión *groovy* de ROS (a principio de 2013) y tiene la capacidad de ser configurable y accesible en tiempo real. La sincronización entre masters se lleva a cabo mediante el paquete *rocon_master_sync* el cual realiza una rutina de exploración y descubrimiento de otros masters que estén en ejecución dentro de la misma red. Su principal restricción es que tan solo es capaz de funcionar entre dos masters por lo que no resulta conveniente si se desea trabajar con más de dos robots.

  - *Fkie multimaster*. Este paquete multimaster es el más completo y el único que se ha mantenido actualizado hasta la actual versión de ROS disponible. Está compuesto por tres nodos, entre los que destacan: el nodo *master_discovey*, encargado de descubrir masters dentro de la misma red sobre el que se ejecuta y el nodo *master_sync* el cual sincroniza el host donde reside con algún otro máster o masters de la red. Los topics, servicios y parámetros de los masters remotos se sincronizan en el máster local haciendo transparente el cambio o la comunicación entre masters. El tercer nodo disponible en el paquete es el *node_manager* el cual ofrece una interfaz con las herramientas que brinda el paquete completo.

    El funcionamiento de este paquete se muestra en la figura 2.22, donde se puede observar que cada robot ejecuta el nodo *ROS MASTER* y el nodo *master_discovery* el cual constantemente envía información a la red para descubrir nuevos masters. El robot 2 además ejecuta el nodo *master_sync* de modo que al hacerlo obtiene todos los topics, parámetros y servicios que se están publicando en los robots 1 y 3, pudiendo suscribirse a cualquiera de ellos de igual modo que lo haría de forma local.

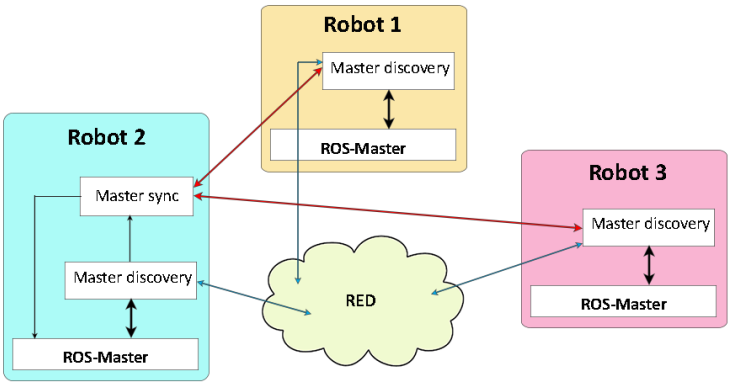


Figura 2.22: Ejemplo de conexiones entre robots con el paquete *fkie_multimaster*.

> La principal desventaja del uso de este paquete es que toda la información de los masters queda expuesta a toda la red, no es posible restringir su visibilidad de manera parcial. La sincronización de los servicios se realiza mediante *polling*, por lo que no se actualiza la información en tiempo real y además el servicio del nodo *master_discovery* satura la red de un modo excesivo ya que constantemente realiza envíos de información mediante broadcast para poder detectar los masters remotos conectados.

Ninguna de las opciones ofrecidas por ROS para su integración con sistemas multi-robot resulta adecuada principalmente por dos razones: la frecuencia de las comunicaciones y la carencia, tanto de protocolos de conversación como de herramientas para incluir información semántica en los mensajes que se envían.

En cuanto a la frecuencia de las comunicaciones, cualquiera de las tres opciones disponibles explicadas anteriormente, pretende mantener la misma filosofía de comunicación que ROS realiza en un nivel local mediante topics y servicios, pero aplicada a un nivel global entre plataformas. Hay que tener en cuenta que ROS consigue llevar a cabo un alto nivel de modularidad gracias su filosofía de definir un nodo como una unidad mínima de ejecución, y esto promueve el interés de crear el mayor número de nodos posibles dentro del sistema siempre que cada uno mantenga un mínimo de relevancia e independencia. El hecho de individualizar los procesos de un sistema robótico no es sencillo puesto que existen muchas dependencias en cuanto a la información que recibe el sistema, su procesamiento y la actuación resultante con la que el sistema debe responder. Este alto nivel de dependencia entre procesos va acorde con el número de topics definidos en el sistema, ya que cuantos más nodos haya en el sistema, mayor dependencia entre nodos existirá y por tanto mayor información habrá que transmitir entre los mismos a través de los topics y servicios. Todo este flujo de información entre nodos, generalmente se lleva

a cabo mediante topics que se publican de manera periódica a altas frecuencias. Cuando se trabaja en un sistema local, esta comunicación es muy rápida y no existe prácticamente ningún retardo, pero obviamente cuando en lugar de una comunicación local, los topics deben atravesar un canal de comunicación más lento como puede ser Wi-Fi u otro protocolo inalámbrico, los retardos pueden suponer un grave problema de sincronización entre sistemas. Es por este motivo por lo que esta metodología no resulta adecuada en arquitecturas donde las comunicaciones entre masters o plataformas no son excesivamente rápidas o simplemente se pretende reservar ancho de banda en el canal de comunicación.

Otra deficiencia importante es tanto la carencia de protocolos de conversación entre sistemas o robots como la falta de información semántica dentro de los mensajes. Los topics y servicios no incorporan facilidades para llevar a cabo conversaciones o protocolos de consenso o toma de decisiones. La alta frecuencia de publicación de los topics hace que su uso para llevar a cabo conversaciones no resulte adecuado y la simplicidad de los servicios por eventos puede hacer tediosa la programación de protocolos complejos con múltiples participantes.

JADE incorpora herramientas y protocolos basados en la especificación FIPA-ACL que facilitan la gestión de las comunicaciones de una forma que ROS hoy en día no es capaz de ofrecer. Estas carencias en la gestión de sistemas multi-robot con ROS y el aprovechamiento de las capacidades de JADE para suplirlas, son la principal motivación que ha promovido el desarrollo del middleware JADE-ROS (en adelante *J-R*) desarrollado en el presente trabajo.

El middleware *J-R* se sitúa entre el RSF ROS y la plataforma multi-agente JADE. ROS queda enmarcado en un nivel inferior y local donde cada robot gestiona sus procesos aprovechando las ventajas que otorga este RSF, y JADE queda situado a un nivel superior y gestiona sobre todo las conversaciones entre los robots y sus mensajes. Esta ha sido la contextualización que se ha definido en las distintas demostraciones desarrolladas de este trabajo, sin embargo el middleware no limita la comunicación directa entre dos sistemas ROS o el uso adicional de alguno de los paquetes multi-master descritos anteriormente.

En la arquitectura desarrollada, cada robot incorporado al sistema multi-robot debe ejecutar el middleware *J-R* para poder comunicarse con el resto de robots del sistema multi-agente. El middleware ejecuta un nodo de ROS el cual incorpora el método de lanzamiento local de un contenedor dentro de la plataforma JADE. Este *nodo-agente* es el encargado de representar al robot dentro de la arquitectura multi-robot y el gestor de cualquier comunicación externa que se requiera como muestra la figura 2.23.

De este modo, se aprovechan las herramientas comunicativas de ROS (topics y servicios) para el desarrollo y la funcionalidad local de cada robot, mientras

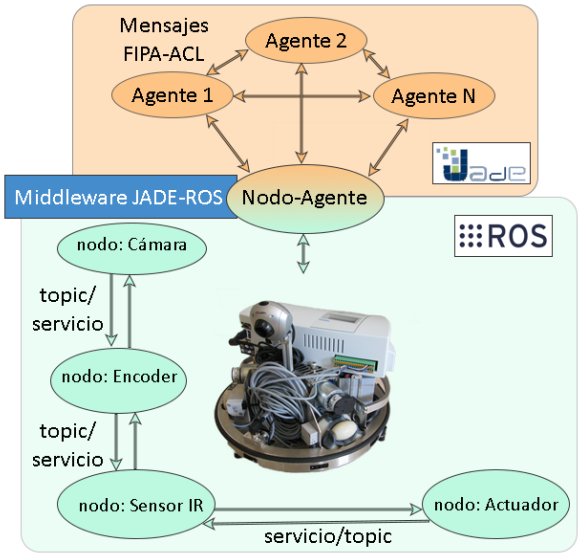


Figura 2.23: Esquema de la arquitectura desarrollada a través del middleware *J-R*

que los protocolos conversacionales FIPA-ACL de JADE, se utilizan para las comunicaciones externas entre robots.

Gracias al middleware *J-R* desarrollado, la integración de cualquier sistema con ROS puede formar parte de un sistema multi-agente basado en JADE donde diferentes robots con otros RSF también pueden formar parte. Además, el administrador o programador de las comunicaciones puede centrarse en la investigación de la interacción entre agentes de una manera totalmente transparente y sin preocuparse por el tipo de lenguaje de programación de cada robot representado.

2.4.3 Aportaciones al software de simulación de robots V-REP

V-REP incorpora, junto con su versión de licencia educacional y libre distribución, unas librerías con modelos 3D de distintos objetos y robots. Entre ellos se incluyen varios modelos de robots móviles como por ejemplo el *khepera* 3, el robot *K-Junior* e incluso el robot *e-puck* utilizado para algunos experimentos prácticos de esta tesis. Con la intención de poder hacer uso de las ventajas que ofrece este simulador, se han desarrollado los modelos de los robots *LEGO Mindstorms NXT* y *Summit XL*.

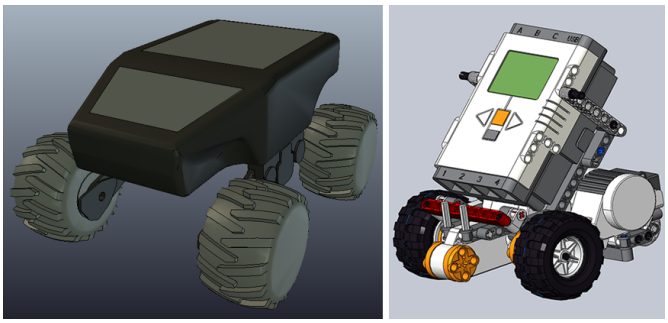


Figura 2.24: Modelos 3D del robot móvil Summit XL (izquierda) y LEGO Mindstorms NXT (derecha)

La integración de un nuevo robot en V-REP se puede dividir principalmente en tres fases. La primera requiere obtener un modelo fiable en 3D del robot en cuestión, de modo que se reconozca visualmente y sea fiel al robot real pero sin embargo no se componga de una malla de polígonos muy pesada, ya que en tal caso, el rendimiento de la simulación en V-REP se ve mermado drásticamente. Dado que el *LEGO Mindstorms* se compone de piezas y módulos que permiten construcciones muy diversas, para este caso se ha escogido un modelo diferencial compuesto por dos ruedas motrices y una rueda castor, el cual es el más utilizado en los demostradores de este trabajo. En la figura 2.24 se muestran los modelos 3D tanto del *Summit XL* como del *LEGO Mindstorms* desarrollado.

La segunda fase requiere incorporar al robot los elementos de los que dispone V-REP para simular el movimiento y la detección de colisiones. Entre las distintas herramientas disponibles para la simulación del movimiento se encuentran por ejemplo torques de fuerza, actuadores, juntas de rotación o engranajes. Para cada rueda, tanto del *LEGO Mindstorms* como el *Summit XL*, se ha añadido en sus ejes, unas juntas de revolución de modo que unen el modelo de cada rueda con el chasis del robot, como muestra la figura 2.25 izquierda.

Posteriormente se recubrió cada modelo con primitivas geométricas que facilitan el cálculo del algoritmo de detección de colisiones de V-REP, también se definió la cinemática y el peso de cada elemento que compone cada robot ya que el simulador tiene en cuenta esos parámetros para ofrecer una simulación más realista. Y por último, en la última fase, se agregaron dispositivos externos como el sensor láser Hokuyo en el caso del *Summit XL*, cuyo modelo ya se encontraba disponible en las librerías de V-REP. La figura 2.25 derecha muestra el modelado terminado del robot *Summit XL* y en la dirección `/https://www.youtube.com/watch?v=kYBb9d6RjkA` o capturando el código QR de la derecha, se puede visualizar un vídeo de ejemplo de la ejecución de la simulación en V-REP.

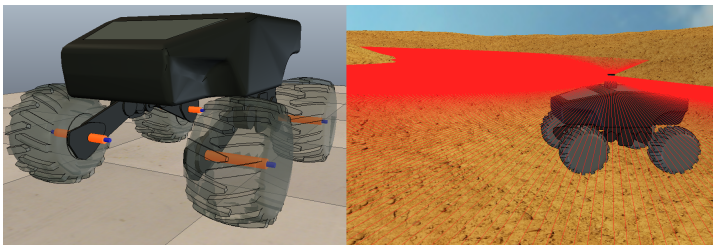


Figura 2.25: A la izquierda se muestra la integración de las juntas de revolución en cada rueda del robot. A la derecha, el modelo 3D con el láser Hokuyo, simulado en V-REP.

Cabe destacar que el modelo desarrollado para esta tesis del robot *Summit XL*, se envió a la empresa Coppelia Robotics y tras algunas modificaciones se incluyó en las librerías predeterminadas que trae consigo el programa V-REP desde su versión 3.1.3 lanzada en octubre de 2014.

2.4.4 Integración JADE y V-REP

Otra de las aportaciones del presente capítulo es la integración del simulador V-REP junto con la plataforma multi-agente JADE con el fin de disponer de un simulador donde previsualizar los demostradores antes de aplicarlos sobre robots reales.

Aprovechando el framework *Remote API* que ofrece V-REP, es posible controlar una simulación a través de aplicaciones externas o de un hardware remoto (un robot real, un computador remoto, etc.). Las funciones de *Remote API* interactúan con V-REP mediante comunicación por sockets, donde cada socket es gestionado por un hilo de ejecución de modo transparente para el usuario, ofreciendo un sistema totalmente distribuido y reduciendo el retardo y la carga de procesamiento. Esta metodología es ideal para trabajar con MAS donde cada agente puede comunicarse individualmente con V-REP y viceversa.

La funcionalidad del framework se divide en dos entidades que interactúan mediante la comunicación socket:

- El lado del servidor (V-REP), el cual debe iniciar un gestor de sockets de comunicación el cual esperará las conexiones pertinentes.
- Y el lado del cliente (Aplicación), que en este caso, se ha desarrollado dentro de cada agente del MAS JADE, a modo de procedimiento de conexión a V-REP mediante un socket.

Para llevar a cabo la comunicación por sockets es necesario configurar V-REP para que ejecute la funcionalidad de servidor mediante alguno de los métodos propuestos en `/http://coppeliarobotics.com/helpFiles/en/remoteApiServerSide.htm`. En el lado del cliente, es decir, en el agente, es necesario incorporar las librerías que ofrece V-REP para poder utilizar las funciones específicas de la *Remote API*, las cuales permiten tanto personalizar la simulación como obtener una realimentación de ella.

En los demostradores desarrollados se ha trabajado siempre con robots móviles por lo que el cómputo y la estrategia del MAS se gestionaba mediante JADE y sin embargo los resultados del módulo de control de cada robot (velocidades de las ruedas, posición y orientación, etc) se envían a V-REP para poder aplicarlos al robot en cuestión y visualizar su comportamiento.

De esta forma, cada agente del MAS incorpora la capacidad de conectarse a V-REP en caso de estar disponible, e iniciar una simulación donde cada robot simulado en V-REP representa un agente del MAS.

2.5 Conclusiones del Capítulo

En este capítulo se ha descrito el estado del arte tecnológico de las arquitecturas distribuidas orientadas al área de la robótica. Además se han expuesto las aportaciones y contribuciones de la presente tesis al desarrollo del MAS JADE así como a los RSFs de ROS y LeJOS, y al software de simulación V-REP:

- Se han desarrollado nuevas herramientas de depuración y monitorización de la plataforma JADE que suplen carencias surgidas a la hora de trabajar con un gran número de agentes dinámicos conversando entre sí para llevar a cabo negociaciones o acuerdos.
- Se han presentado las posibilidades ofrecidas por los RSFs de LeJOS, ROS y del software de simulación 3D V-REP a la hora de trabajar con sistemas multi-robot, así como sus limitaciones.
- Se ha descrito la necesidad y el desarrollo de un middleware de control y comunicaciones capaz de integrar el MAS JADE con uno de los RSFs más extendidos para la programación del robot Lego Mindstorms, el firmware LeJOS. Se han especificado las ventajas ofrecidas por el middleware *J-L* y la posibilidad de utilizarlo para su integración dentro de las distintas versiones del robot LEGO, entre las que destaca su reciente versión EV3.
- Se ha presentado el middleware de control y comunicación *J-R* desarrollado, capaz de integrarse en sistemas robóticos que trabajan bajo la extendida arquitectura distribuida ROS; pudiendo así formar parte de un sistema multi-

agente basado en JADE que hace uso de las ventajas de los protocolos de comunicación basados en la especificación FIPA-ACL.

- Se han introducido nuevos modelos de los robots LEGO Mindstorms y Summit XL en el software de simulación V-REP. Dichos modelos han sido creados digitalmente en 3D y posteriormente se les ha dotado de sus especificaciones físicas reales de masa, rozamiento, cinética, etc. para su correcta simulación.
- Se ha integrado el MAS JADE con el software V-REP pudiendo de ese modo realizar simulaciones del movimiento de los agentes robóticos evitando la problemática inherente a las plataformas y experimentos reales.
- Los diferentes middlewares desarrollados ofrecen una transparencia absoluta entre la capa de gestión del sistema multi-agente y la programación específica de cada robot individual, consiguiendo así la homogeneidad del sistema que se pretendía.

Capítulo 3

Autolocalización Local mediante Filtros de Fusión Sensorial con Corrección Continua

Tal y como se comentó en la introducción de esta tesis, la localización juega un papel fundamental en sistemas donde un grupo de robots móviles heterogéneo comparte un escenario en la realización de sus misiones individuales o conjuntas. Además, un buen conocimiento del posicionamiento de los robots puede resultar muy útil e incluso decisivo para utilizar dicha localización como entrada a un algoritmo posterior de navegación con planificación de trayectorias o de evasión de colisiones.

La problemática de la tarea de localización es un tema ampliamente estudiado por investigadores del área en trabajos como [163], [167], [173], [43] o [5]. Existen principalmente dos estrategias de auto-localización: algoritmos de localización globales que trabajan a través del reconocimiento del entorno o escenario sin conocimiento de su posición o orientación inicial (estimación de posición global) y algoritmos de seguimiento de posición que parten del conocimiento de la posición inicial del robot al comienzo de la ejecución (también conocida como estimación de la posición local).

La mayoría de las estrategias de seguimiento de posición que trabajan con el conocimiento de una posición inicial conocida, plantean el uso de filtros de Kalman ([72], [190]) para llevar a cabo la fusión de medidas recogidas por sensores embarcados en el robot con el fin de mejorar significativamente los problemas de odometría y así otorgar a la auto-localización una mayor eficacia y precisión. Cuando se trabaja con sistemas de recursos limitados, los sensores disponibles para embarcar

en estos sistemas suelen ser de bajo coste y la resolución en sus medidas, de baja precisión. Sin embargo, con el correcto filtro de fusión de datos, es posible mejorar considerablemente la estimación de su posición. En este tipo de sistemas se pueden implementar filtros de fusión simples o reducidos cuyo cómputo es soportable por el procesador. Es bien conocido que este tipo de estimación local, a pesar de mejorar la estimación de posición notablemente, siempre contiene pequeños errores que, debido a su realimentación dentro del sistema, tienden a aumentar progresivamente con el paso del tiempo de ejecución, perjudicando paulatinamente la estimación. Dentro de las distintas estrategias de resolución de este problema, se encuentra la actualización de la posición global mediante un supervisor o supervisores externos, las cuales se plantean en posteriores capítulos.

La columna vertebral de los métodos de estimación de posición local es la *navegación por estimación* o, en inglés conocido como el procedimiento de *dead reckoning*. Esta técnica intenta inferir la ubicación de un robot móvil haciendo uso de cálculos basados en el rumbo y la velocidad de navegación a lo largo de un cierto periodo de tiempo. La entrada a este tipo de estrategias puede ser cualquier tipo de medida relativa al movimiento del robot. El conjunto de dichas medidas se fusiona con la intención de obtener una estimación de posición mejorada, por lo que no sólo la sensorización embarcada en el robot juega un papel fundamental en la precisión de este método, sino que también la metodología utilizada para realizar tal fusión de datos.

En este capítulo se describe la implementación de filtros de fusión sensorial aplicada a robots móviles de recursos limitados para la mejora de la estimación local de posición.

3.1 Especificación del Problema de la Fusión Sensorial

Formalmente, este problema trata de controlar un proceso dinámico observable y controlable donde, dado un estado x_k afectado por un ruido w_k, se pretende obtener una salida y_k próxima a una referencia r_k dada, haciendo uso de una sensorización medida z_k con un ruido n_k. Esto se reduce al diagrama común de un lazo de control como el que se muestra en la figura 3.1, donde el controlador calcula el error e_k como la diferencia entre la medición z_k y la referencia deseada r_k.

En los casos donde no es posible obtener una medición directa o indirecta de todos los procesos que se desean controlar, se modifica el lazo de control introduciendo un estimador. En general, los métodos de estimación buscan obtener el valor de todos los estados de un sistema observable [166] o de ciertos estados de interés sin posibilidad de sensorización, al utilizar las mediciones de las entradas u_k y las salidas del sistema, medidas a través de uno o varios sensores, z_k.

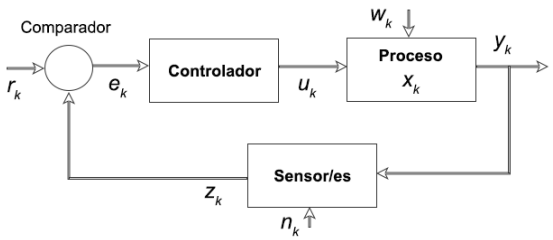


Figura 3.1: Bucle de control realimentado por uno o varios sensores.

Los estimadores requieren la definición de un modelo del sistema fiable y, para algunos métodos, información sobre las propiedades estadísticas de los ruidos w_k y n_k. Tal y como muestra la figura 3.2, el estado estimado $\hat{x_k}$ es utilizado como entrada al controlador para minimizar e_k mediante el regulador diseñado. Cuando el error de estimación $(x_k - \hat{x_k})$ es acotado o es cero, se puede considerar que la estimación del estado es óptima.

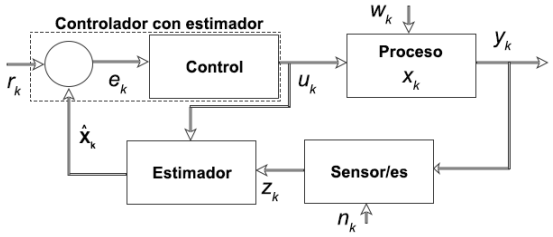


Figura 3.2: Bucle de control realimentado por uno o varios sensores con estimador de estado.

Con el objetivo de simplificar la notación en los algoritmos y ecuaciones, se considera un modelo *lineal* cuando puede escribirse tal y como se muestra en la ecuación 3.1, con el vector de entrada $u_k \in \Re^u$, el vector de medición $z_k \in \Re^m$ y el vector de estados $x_k \in \Re^n$ además de la matriz de estados A_k, de entradas B_k y de medición H_k (constantes en caso de sistemas invariantes en el tiempo). Por otro lado, si el modelo se puede representar mediante la ecuación 3.2, se considera un modelo *no lineal*, que requiere una función no lineal f para relacionar el estado x_k en el instante k con el estado en el instante anterior $k-1$, y una función no lineal h que relaciona x_k con z_k.

$$\begin{aligned} \mathbf{x_k} &= \mathbf{A_{k-1}}\mathbf{x}_{k-1} + \mathbf{B_{k-1}}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \\ \mathbf{z}_k &= \mathbf{H_k}\mathbf{x}_k + \mathbf{n}_k \end{aligned} \tag{3.1}$$

$$\begin{aligned} \mathbf{x}_k &= f\left(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{w}_{k-1}\right) \\ \mathbf{z}_k &= h\left(\mathbf{x}_k, \mathbf{n}_k\right) \end{aligned} \tag{3.2}$$

En ambas ecuaciones se definen los términos w_k y n_k correspondientes al ruido del proceso y de la medición, respectivamente, tal y como se define en [166].

El tipo de estimadores del esquema de la figura 3.2, requieren la existencia de observadores de estado del sistema que tengan en cuenta la presencia del ruido del proceso w_k y de medición n_k, al considerar un modelo estocástico del sistema como es el caso de las ecuaciones 3.1 y 3.2. Estos observadores son filtros aplicados a la estimación de estados que requieren: diseñar la ganancia K_k para incorporar el error de estimación, un método para administrar el modo en el que se agrega la información asociada a (w_k, n_k) en el observador (mediante K_k), y una estimación del efecto de (w_k, n_k) en el sistema según su evolución con el paso del tiempo. La metodología con la que se realizan estas tareas difiere según el filtro de estimación que se utilice, sin embargo, generalmente se recurren a herramientas probabilísticas o estadísticas para describir las propiedades de w_k y n_k y a técnicas de optimización para la obtención de K_k con la finalidad de minimizar algún tipo de índice de desempeño [166].

Los algoritmos que implementan filtros de fusión de datos suelen utilizar métodos recursivos de programación, dividiendo su ejecución en distintas etapas, tal y como se muestra a continuación en la figura 3.3.

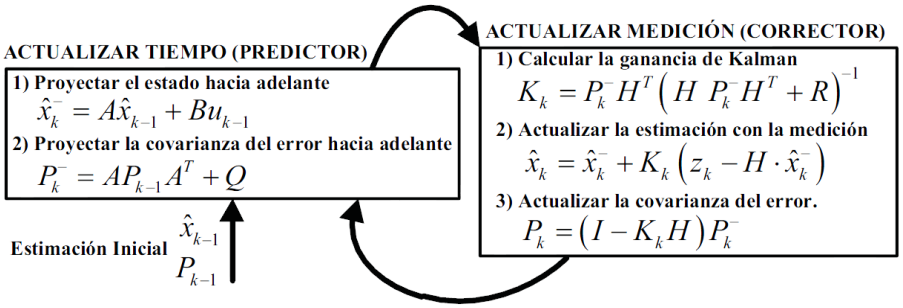


Figura 3.3: Estructura Predicción/Corrección en el Filtro de Kalman [191]

En la etapa de predicción se utiliza el modelo para actualizar su estimación actual teniendo en cuenta el instante anterior, y en la etapa de corrección se incorpora a la estimación la información recibida desde los sensores (la medición).

Entre la literatura ([191], [86], [20]) existen multitud de filtros de datos distintos según el tipo de sistema sobre el cual se desea aplicar. Entre ellos cabe destacar los siguientes:

- Filtros de estimación para sistemas lineales:
  - Filtro de Kalman (KF). Se trata del filtro por excelencia en sistemas lineales. El KF busca estimar el estado x_k considerando que tanto w_k

como n_k (ruidos del proceso y de medición) se pueden caracterizar mediante una distribución de probabilidad independiente, blanca y normal (Gaussiana) con media cero, [166]. La estimación se lleva a cabo a partir de las matrices A, B y H del modelo 3.1, de las matrices de covarianza Q_k y R_k correspondientes a los ruidos w_k y n_k respectivamente, y de las entradas u_k y mediciones z_k que se realizan. Con dichas entradas, el filtro estima el estado $\hat{\mathrm{x}}$ como la covarianza P_x correspondiente a dicha estimación, $\epsilon_{x,k} = (x_k - \hat{x}_k)$.

- Filtro H_∞. Este filtro es conocido por ser una alternativa al KF aplicable a sistemas lineales. En este caso, en lugar de calcular los índices de coste como los valores mínimos de las varianzas, este filtro busca minimizar el peor caso del error de estimación mediante estrategias de optimización como *minmax*, cuya obtención se puede realizar mediante multiplicadores de Lagrange y otras estrategias de teoría de juegos [166]. Otra diferencia notable respecto al KF es que el Filtro H_∞ considera tanto los errores de modelado (matrices A, B, H), como los de la especificación del ruido del proceso y de la medida (matrices Q y R) desde el momento de su diseño. Este filtro no restringe los tipos de ruido sobre los que puede trabajar por lo que es aplicable a un mayor número de sistemas. No obstante, su implementación es más compleja y consume gran cantidad de recursos computacionales.

- Filtros de estimación para sistemas no lineales: A diferencia de los sistemas lineales, los sistemas no lineales continúan suponiendo un reto importante dentro del control, la predicción y la estimación. A pesar de encontrarse en constante desarrollo, a continuación se exponen los métodos más relevantes en filtros de estimación no lineal a fecha de hoy:

  - Filtro Extendido de Kalman (EKF). El EKF utiliza una aproximación del modelo del sistema a partir de su linealización mediante series de Taylor de primer orden. De ese modo, mediante dicha linealización, es viable aplicar la misma estrategia que el KF utiliza para sistemas lineales. La complejidad de su implementación depende de la no-linealidad de los parámetros del sistema.[[166]].

  - Filtros con Aproximaciones de la Densidad pdf. Este tipo de filtros intentan evitar o menguar los errores producidos por la linealización del EKF en sistemas no lineales. Debido a que el EKF tan sólo utiliza las series de Taylor de primer orden, la representación del sistema puede resultar poco exacta y por ello este tipo de filtros requieren del cálculo de términos adicionales de la aproximación de Taylor para obtener mejor precisión. Estos cálculos suelen suponer un alto coste computacional y temporal, por lo que no resultan adecuados para sistemas rápidos o con un tiempo de ciclo corto. [[29],[77]].

- Filtros de Partículas (PF). Los métodos de filtrado por partículas o métodos de Monte Carlo (SMC) son aplicados en sistemas altamente no lineales donde no es posible llevar a cabo ninguna linealización y por tanto, los KF no ofrecen ninguna solución. Las partículas corresponden a estados posibles que se van propagando a través del sistema y al mismo tiempo van calculando las covarianzas asociadas a dichos estados. El proceso va eliminando las partículas de baja probabilidad de éxito y duplicando las mejores candidatas o en algunos casos introduciendo estrategias de adición de mutaciones o incluso estados al azar. [[53], [52]].

3.1.1 Selección de Algoritmos de Fusión Sensorial

Una vez expuestos los distintos filtros y estrategias de fusión sensorial, se debe seleccionar una técnica que sea aplicable a los distintos tipos de hardware sobre los cuales se va a trabajar. En la mayoría de casos de esta tesis, se trata de procesadores de bajas prestaciones o recursos limitados. Además, conviene que la estrategia escogida ofrezca cierta flexibilidad dado que se trabaja con tipos de robots heterogéneos equipados con distintos tipos de sensores.

En este sentido, las técnicas de estimación basadas en observadores requieren menos recursos computacionales (en cuanto a memoria y tiempo de ejecución) que las basadas en filtros, como se justifica en [82], no obstante requieren modelos muy precisos del sistema para funcionar correctamente, dado que no tienen en cuenta ruidos en el proceso ni en la medición.

Por otra parte, las técnicas de estimación basadas en filtros sí que consideran los ruidos en el proceso y en la medición, y son escalables en requerimientos computacionales en función del nivel de precisión que se pueda requerir. Además mediante el uso de dichas técnicas es posible tener en cuenta la precisión relativa a cada sensor (matriz R_k) y su aplicación en casos no lineales resulta más sencilla de implementar.

Por estos motivos, la integración de algoritmos de fusión sensorial se ha llevado a cabo a través de estrategias basadas en filtros de estimación. Se debe tener en cuenta que estos filtros requieren normalmente una o varias operaciones matriciales complejas que no son sencillas de implementar en sistemas con recursos limitados, como son el cálculo de la inversa de una matriz o su raíz cuadrada. Además, la dimensión de dichas matrices no es fija, sino que depende de la cantidad de estados n que se desean estimar en el sistema así como el número de sensores considerados. Esta escalabilidad de las dimensiones de las matrices dificulta su implementación más aún en este tipo de sistemas, y repercute en la necesidad de tener que aplicar estrategias de reducción de código, reutilización de variables y de ajuste de tiempos de ejecución. Es por este motivo por el que se ha escogido para este trabajo la

implementación de técnicas de fusión basadas en el KF para los modelos lineales y el EKF para los modelos no lineales. En ambos casos, se ha descartado la idea de utilizar una matriz de ganancias constante a pesar del aumento de coste computacional que ello conlleva, con la finalidad de poder aprovechar el aporte de la lectura de los sensores en tiempo real (matriz R_k) y explotar las ventajas del uso de estos filtros de fusión sensorial.

A continuación se describe el desarrollo del primer algoritmo de fusión sensorial implementado para este trabajo.

3.2 Metodología del Algoritmo de Fusión Sensorial de Corrección Continua

Cuando se habla de corrección continua, se hace referencia a que en cada ciclo del tiempo de muestreo (T_s) o iteración del bucle de control del robot, se realiza una ejecución completa del algoritmo de filtrado y se estima una nueva *pose* o *postura* del robot. En plataformas de recursos limitados esto supone todo un reto dado que en un solo tiempo de ciclo se deben obtener los valores de todos los sensores que se utilizan como entrada, realizar el filtrado, la fusión de los mismos, y obtener como salida finalmente la *pose* resultante.

El desarrollo y la implementación de los algoritmos de fusión sensorial de corrección continua de esta tesis se han llevado a cabo en dos fases. La primera fase tuvo como objetivo reducir el coste de ejecución teórico y práctico de este tipo de algoritmos para llegar a integrarlos en plataformas con una escasa capacidad de cómputo. La segunda se centró en la incorporación de estos filtros a sistemas de control distribuidos donde cada proceso o agente gestiona cada recurso y se requiere de una comunicación coordinada para reunir toda la información necesaria para su ejecución. A continuación se describen los procedimientos empleados para abordar ambos hitos.

3.2.1 Integración de Algoritmos de Fusión en Plataformas de Recursos Limitados

Para el desarrollo de los algoritmos de fusión sensorial se utilizan los modelos de las versiones cinemáticas y dinámicas de las dos configuraciones definidas en el anexo B. Concretamente las ecuaciones (B.11) y (B.24) para la configuración diferencial y (B.36) y (B.51) para la configuración Ackerman. Estos modelos se usan junto con la entrada $\boldsymbol{u}_k \in \Re^u$ y con las mediciones locales ($\boldsymbol{z}_k \in \Re^m$) obtenidas de los sensores embarcados en las plataformas experimentales para, finalmente, estimar el estado definido como $\boldsymbol{x}_k \in \Re^n$.

Para la obtención de la localización del robot móvil se define el vector de medición $z_k = H_k x_k$ formado por las mediciones útiles de los sensores embarcados. Estas mediciones comprenden las velocidades lineales $v_{x,enc}$, $v_{y,enc}$ y la angular w_{enc} calculadas a partir de la lectura de los encoders de los motores. Además, incluyen también los valores ω_{gyr} y ω_{comp} medidos mediante un giróscopo y una brújula respectivamente, junto con las aceleraciones (u_1, u_2, u_3) obtenidas mediante dos acelerómetros 3D colocados según el modelo de partículas presentado en [116] y [115], resumido en el anexo B (modelos (B.23),(B.50), figuras B.4,B.6), donde u_1 es la suma de las aceleraciones en el eje x local, u_2 la suma para el eje y y u_3 la diferencia de las aceleraciones en el eje y (según la ecuación (B.49)). Adicionalmente, como condición inicial, se debe conocer la *pose* del robot respecto a las coordenadas globales del escenario. Esta *pose* es una medida adicional de medición cuyo valor irá actualizándose conforme se produzca el desplazamiento del vehículo.

Hay que tener en cuenta que el modelo dinámico de la configuración diferencial ((B.24)) no considera deslizamiento por lo que el vector z_k es más reducido ya que tanto la velocidad como la aceleración lateral son nulas, es decir, $v_{y,enc} = 0$ y $u_2 = 0$.

De este modo, el vector de mediciones se define según la ecuación (3.3) para el caso general y según la ecuación (3.4) para el caso de no deslizamiento en el eje y local.

$$\boldsymbol{z}_k = H_k x_k = \begin{bmatrix} v_{x,enc} & v_{y,enc} & \omega_{enc} & \omega_{gyr} & \omega_{comp} & u_1 & u_2 & u_3 & x & y & \theta \end{bmatrix}^T \tag{3.3}$$

$$\boldsymbol{z}_k = \begin{bmatrix} v_{x,enc} & \omega_{enc} & \omega_{gyr} & \omega_{comp} & u_1 & u_3 & x & y & \theta \end{bmatrix}^T \tag{3.4}$$

Al utilizar un filtro de estimación como el EKF o el UKF, el resultado que se logra obtener es la postura estimada del robot móvil $\hat{\boldsymbol{x}}_k = [x, y, \theta]_k$ y la covarianza del correspondiente error de estimación P_k, en cada instante de muestreo k.

Como entradas iniciales, el algoritmo requiere conocer el estado x_0 y la covarianza inicial P_0 (pose del robot al inicio del movimiento y su covarianza asociada) junto con las matrices Q_k y R_k del ruido del proceso y el de la medición respectivamente. La matriz de ruido R_k puede ajustarse inicialmente como una matriz diagonal con las covarianzas σ^2 individuales de cada sensor y la matriz Q_k como otra matriz diagonal con las covarianzas iniciales de cada estado.

El EKF realiza la fusión a través de a matriz H_k obtenida mediante la linealización de la ecuación de medición del modelo en el caso no lineal, o bien mediante la definición lineal de las ecuaciones (3.3) y (3.4). Esta matriz H_k establece una ecuación por cada estado e incorpora las mediciones a la estimación mediante la ganancia K_k, la cual depende de la precisión de cada sensor, definida en la matriz R_k. El algoritmo 1 describe de manera esquemática el funcionamiento del EKF.

Algoritmo 1: Algoritmo EKF con corrección continua

Entrada:
u_{k-1},$\hat{x}_{k-1}$,P_{k-1}

Medición:
z_k=$\left[\, v_{x,enc} \;\; v_{y,enc} \;\; \omega_{enc} \;\; \omega_{gyr} \;\; \omega_{comp} \;\; u_1 \;\; u_2 \;\; u_3 \;\; x \;\; y \;\; \theta \,\right]^T$

Datos:
$f(\cdot)$ *y* $h(\cdot)$ *(del modelo no lineal* (B.24) *o* (B.51)*)*,Q_k,R_k

Salida:
$\hat{x}_k$,P_k

Inicialización: $\hat{x}_0$,P_0

Para *el instante actual* k **hacer**

Predicción:

$\hat{x}_k = f\left(\hat{x}_{k-1}, u_{k-1}, 0\right)$

$A_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1},u_{k-1},0}$, $W_k = \left.\frac{\partial f}{\partial w}\right|_{\hat{x}_{k-1},u_{k-1},0}$

$P_k = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$

Corrección:

$H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_k,0}$, $N_k = \left.\frac{\partial h}{\partial n}\right|_{\hat{x}_k,0}$

$K_k = P_k H_k^T \left(H_k P_k H_k^T + N_k R_k N_k^T\right)^{-1}$

$\hat{x}_k = \hat{x}_k + K_k \left[z_k - h\left(\hat{x}_k, 0\right)\right]$

$P_k = \left(I - K_k H_k\right) P_k$

fin

En el caso del UKF (algoritmo 2) el procedimiento es muy similar, pero se utiliza la propagación de los puntos sigma para obtener z_k.

Algoritmo 2: Algoritmo UKF con corrección continua

Entrada: u_{k-1},$\hat{x}_{k-1}$,P_{k-1}

Medición: z_k=$\left[v_{x,enc}\ v_{y,enc}\ \omega_{enc}\ \omega_{gyr}\ \omega_{comp}\ u_1\ u_2\ u_3\ x\ y\ \theta\right]^T$

Datos: $f(\cdot)$ *y* $h(\cdot)$ *(del modelo no lineal* (B.24) *o* (B.51)*)*,Q_k,R_k,n,κ

Salida: $\hat{x}_k$,P_k

Inicialización: Ampliar $\hat{x}_0^{(a)} = E\left[x, w, n\right]_k^T$ y $P_0^{(a)} = \text{diag}\{P_0, Q_0, R_0\}$, Definir Pesos: $W_{(0)} = \kappa/(n+\kappa)$,$W_{(i)} = 1/2\,(n+\kappa), i = 1, \cdots, 2n$

Para *el instante actual* k **hacer**

Predicción:

Obtener puntos sigma $\hat{x}_{k-1}^{(i)}$:

$\hat{x}_{k-1}^{(0)} = \hat{x}_{k-1}$

$$\hat{x}_{k-1}^{(i)} = \hat{x}_{k-1} + \widetilde{\mathrm{P}}_{(i)} \begin{cases} \widetilde{\mathrm{P}}_{(i)} = \left(\sqrt{(n+\kappa)\,P_{k-1}}\right)_{(i)}, i = 1, \cdots, n \\ \widetilde{\mathrm{P}}_{(i)} = -\left(\sqrt{(n+\kappa)\,P_{k-1}}\right)_{(i)}, i = n+1, \cdots, 2n \end{cases}$$

$$\hat{x}_k^{(i)} = f\left(\hat{x}_{k-1}^{(i)}, u_{k-1}\right), i = 0, \cdots, 2n$$

$$\hat{x}_k = \sum_{i=0}^{2n} W_{(i)} \hat{x}_k^{(i)}$$

$$P_k = \sum_{i=0}^{2n} W_{(i)} \left(\hat{x}_k^{(i)} - \hat{x}_k\right)\left(\hat{x}_k^{(i)} - \hat{x}_k\right)^T$$

Obtener Puntos Sigma $x_k^{(i)}$

$$\hat{z}_k^{(i)} = h\left(\hat{x}_k^{(i)}\right), i = 0, \cdots, 2n$$

$$\hat{z}_k = \sum_{i=0}^{2n} W_{(i)} \hat{z}_k^{(i)}$$

$$P_z = \sum_{i=0}^{2n} W_{(i)} \left(\hat{z}_k^{(i)} - \hat{z}_k\right)\left(\hat{z}_k^{(i)} - \hat{z}_k\right)^T$$

$$P_{xz} = \sum_{i=0}^{2n} W_{(i)} \left(\hat{x}_k^{(i)} - \hat{x}_k^-\right)\left(\hat{z}_k^{(i)} - \hat{z}_k\right)^T$$

Corrección:

$$K_k = P_{xz}(P_z)^{-1}$$

$$\hat{x}_k = \hat{x}_k + K_k\left(z_k - \hat{z}_k\right)$$

$$P_k = P_k - K_k P_z K_k^T$$

fin

Una vez expuestos los algoritmos base, el siguiente paso es el planteamiento de estrategias de reducción de coste computacional aplicadas tanto a sistemas no lineales, como a sistemas lineales.

3.2.1.1 Reducción del Coste Computacional de los Algoritmos de Fusión Basados en Modelos Dinámicos no Lineales

Con el fin de reducir el coste de los algoritmos de fusión sensorial, una primera estrategia aplicable es la reducción del número de entradas del modelo. Con este objetivo se plantea utilizar los modelos dinámicos globales no lineales obtenidos por la descomposición de partículas (ecuación (B.24) para el modelo diferencial y ecuación (B.51) para el modelo Ackerman), en los cuales es posible utilizar como entrada las aceleraciones obtenidas de los acelerómetros del robot.

De esta manera se define el vector de entrada u_k según la ecuación (3.5) para el caso general y (3.6) para el caso de no deslizamiento en el eje y de la plataforma.

$$\boldsymbol{u}_k = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}^T \tag{3.5}$$

$$\boldsymbol{u}_k = \begin{bmatrix} u_1 & u_3 \end{bmatrix}^T \tag{3.6}$$

Así pues se reduce el vector de medición z_k tal y como se muestra en las ecuaciones (3.7) para el caso general y (3.8) para el caso sin deslizamiento.

$$\boldsymbol{z}_{k,i} = \begin{bmatrix} v_{x,enc} & v_{y,enc} & \omega_{enc} & \omega_{gyr} & \omega_{comp} & x & y & \theta \end{bmatrix}^T \tag{3.7}$$

$$\boldsymbol{z}_{k,i} = \begin{bmatrix} v_{x,enc} & \omega_{enc} & \omega_{gyr} & \omega_{comp} & x & y & \theta \end{bmatrix}^T \tag{3.8}$$

Gracias a esta reducción la inversión de la matriz de entrada pasa de tener unas dimensiones de 10×10 ó 11×11(caso de no deslizamiento en eje y) a una matriz de 7×7 ú 8×8 por lo que la reducción en términos computacionales resulta conveniente y adecuada.

Haciendo uso de estos nuevos vectores de entrada, se reescriben los algoritmos del EKF (1) y UKF (2) obteniendo dos nuevos algoritmos 3 (EKF) y 4 (UKF), cuyo pseudocódigo se muestra a continuación.

Algoritmo 3: Algoritmo EKF recursivo con reasignación de entradas y corrección continua

Entrada:

$u_{k-1} = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}^T$,$\hat{x}_{k-1}$,$P_{k-1}$

Medición:

$z_k = \begin{bmatrix} v_{x,enc} & v_{y,enc} & \omega_{enc} & \omega_{gyr} & \omega_{comp} & x & y & \theta \end{bmatrix}^T$

Datos:

$f(\cdot)$ *y* $h(\cdot)$ *(del modelo no lineal* (B.24) *o* (B.51)*)*,Q_k,R_k

Salida:

$\hat{x}_k$,P_k

Inicialización: $\hat{x}_0$,P_0

Para *el instante actual* k **hacer**

Predicción:

$$\hat{x}_k = f\left(\hat{x}_{k-1}, u_{k-1}, 0\right)$$
$$A_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1}, u_{k-1}, 0}$$
$$W_k = \left.\frac{\partial f}{\partial w}\right|_{\hat{x}_{k-1}, u_{k-1}, 0}$$
$$P_k = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$$

Corrección:

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_k, 0}$$
$$N_k = \left.\frac{\partial h}{\partial n}\right|_{\hat{x}_k, 0}$$
$$K_k = P_k H_k^T \left(H_k P_k H_k^T + N_k R_k N_k^T\right)^{-1}$$
$$\hat{x}_k = \hat{x}_k + K_k \left[z_k - h\left(\hat{x}_k, 0\right)\right]$$
$$P_k = \left(I - K_k H_k\right) P_k$$

fin

Algoritmo 4: Algoritmo UKF recursivo con reasignación de entradas y corrección continua

Entrada: $u_{k-1} = [\ u_1 \quad u_2 \quad u_3\ ]^T$,$\hat{x}_{k-1}$,$P_{k-1}$

Medición: $z_k = \left[v_{x,enc}\ \ v_{y,enc}\ \ \omega_{enc}\ \ \omega_{gyr}\ \ \omega_{comp}\ \ x\ \ y\ \ \theta\right]^T$

Datos: $f(\cdot)$ *y* $h(\cdot)$ *(del modelo no lineal* (B.24) *o* (B.51)*)*,Q_k,R_k,n,κ

Salida: $\hat{x}_k$,P_k

Inicialización: Ampliar $\hat{x}_0^{(a)} = E\left[x, w, n\right]_k^T$ y $P_0^{(a)} = \text{diag}\{P_0, Q_0, R_0\}$, Definir Pesos: $W_{(0)} = \kappa/(n+\kappa)$,$W_{(i)} = 1/2\,(n+\kappa), i = 1, \cdots, 2n$

Para *el instante actual* k **hacer**

Predicción:

Obtener puntos sigma $\hat{x}_{k-1}^{(i)}$: $\hat{x}_{k-1}^{(0)} = \hat{x}_{k-1}$

$$\hat{x}_{k-1}^{(i)} = \hat{x}_{k-1} + \widetilde{\mathrm{P}}_{(i)} \begin{cases} \widetilde{\mathrm{P}}_{(i)} = \left(\sqrt{(n+\kappa)\,P_{k-1}}\right)_{(i)}, i = 1, \cdots, n \\ \widetilde{\mathrm{P}}_{(i)} = -\left(\sqrt{(n+\kappa)\,P_{k-1}}\right)_{(i)}, i = n+1, \cdots, 2n \end{cases}$$

$$\hat{x}_k^{(i)} = f\left(\hat{x}_{k-1}^{(i)}, u_{k-1}\right), i = 0, \cdots, 2n$$

$$\hat{x}_k = \sum_{i=0}^{2n} W_{(i)} \hat{x}_k^{(i)}$$

$$P_k = \sum_{i=0}^{2n} W_{(i)} \left(\hat{x}_k^{(i)} - \hat{x}_k\right)\left(\hat{x}_k^{(i)} - \hat{x}_k\right)^T$$

Obtener Puntos Sigma $x_k^{(i)}$

$$\hat{z}_k^{(i)} = h\left(\hat{x}_k^{(i)}\right), i = 0, \cdots, 2n$$

$$\hat{z}_k = \sum_{i=0}^{2n} W_{(i)} \hat{z}_k^{(i)}$$

$$P_z = \sum_{i=0}^{2n} W_{(i)} \left(\hat{z}_k^{(i)} - \hat{z}_k\right)\left(\hat{z}_k^{(i)} - \hat{z}_k\right)^T$$

$$P_{xz} = \sum_{i=0}^{2n} W_{(i)} \left(\hat{x}_k^{(i)} - \hat{x}_k^-\right)\left(\hat{z}_k^{(i)} - \hat{z}_k\right)^T$$

Corrección:

$$K_k = P_{xz}(P_z)^{-1}$$

$$\hat{x}_k = \hat{x}_k + K_k\left(z_k - \hat{z}_k\right)$$

$$P_k = P_k - K_k P_z K_k^T$$

fin

La segunda reducción aplicada que se plantea consiste en cambiar la estructura del filtrado para dividirla en dos fases. Una primera fase consiste en un filtrado local donde únicamente se utilizan los sensores locales (las velocidades calculadas a partir de ellos) para actualizar el estado y la covarianza. Y una segunda etapa donde se realiza el filtrado global empleando únicamente los sensores globales (la *pose* global) para actualizar el estado y la covarianza. Estos sensores globales se corresponden con dispositivos de posicionamiento global como es el caso de GPS, cámaras cenitales de posicionamiento en un espacios interiores, códigos QR de señalización, etc. Las variables asociadas a estos dispositivos se han marcado con el subíndice GP como referencia a *Global Position*.

Se realiza de nuevo una reducción del vector de medición z_k teniendo en cuenta que el robot donde se va a implementar este filtro está sensorizado con dos encoders, un giróscopo y una brújula.

$$\begin{aligned}
\boldsymbol{x}_k &= \begin{bmatrix} x & y & \theta & v_x & v_y & \omega \end{bmatrix}_k^T = \begin{bmatrix} x_{k,p} & x_{k,v} \end{bmatrix}^T \\
\boldsymbol{x}_{k,p} &= \begin{bmatrix} x & y & \theta \end{bmatrix}_k^T \\
\boldsymbol{x}_{k,v} &= \begin{bmatrix} v_x & v_y & \omega \end{bmatrix}_k^T \\
\boldsymbol{u}_k &= \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}_k^T \\
z_k &= \begin{bmatrix} \{ \begin{matrix} x & y & \theta \end{matrix} \}_{GP} & \begin{matrix} v_{x,enc} & v_{y,enc} & \omega_{enc} & \omega_{gyr} & \omega_{comp} \end{matrix} \end{bmatrix}_k^T \\
&= \begin{bmatrix} z_{k,p} & z_{k,v} \end{bmatrix}^T \\
\boldsymbol{z}_{k,p} &= \boldsymbol{L}_{GP} = \begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}_{GP}^T \\
\boldsymbol{z}_{k,v} &= \begin{bmatrix} v_{x,enc} & v_{y,enc} & \omega_{enc} & \omega_{gyr} & \omega_{comp} \end{bmatrix}_k^T
\end{aligned} \tag{3.9}$$

Esta reducción mostrada en la ecuación (3.9), permite realizar la división del filtro comentada anteriormente. En este caso en lugar de realizar la inversión de una matriz de 8×8, el filtrado realizará la inversión de dos matrices, una de dimensiones de 5×5 (la local) y otra de dimensiones 3×3. Gracias al ahorro del coste computacional que supone, ha sido posible su implementación en la plataforma objetivo de recursos limitados.

A este tipo de filtros se les denomina filtros en cascada ya que la salida del filtro local $(\hat{x}_k, P_k)$ actualizada con $z_{k,v}$ es utilizada como entrada del filtro global, cuya salida $(\hat{x}_k, P_k)$ es actualizada mediante $z_{k,p}$.

Con el objetivo de seccionar el algoritmo de filtrado para poder realizar su división, se realiza un análisis del EKF en componentes, comenzando por linealizar la *pose*

del robot de configuración Ackerman (ecuación (B.51)) según refleja la ecuación (3.10).

$$\begin{aligned}
\begin{bmatrix} x \\ y \\ \theta \\ v_x \\ v_y \\ \omega \end{bmatrix}_k &= \begin{bmatrix} x \\ y \\ \theta \\ v_x \\ v_y \\ \omega \end{bmatrix}_{k-1} + T_s \begin{bmatrix} v_x\cos\theta - v_y\sin\theta \\ v_x\sin\theta + v_y\cos\theta \\ \omega \\ v_y\omega + \lambda_a u_1 \\ -v_x\omega + \lambda_a u_2 \\ \lambda_\alpha \gamma u_3 \end{bmatrix}_{k-1} \\
&= \begin{bmatrix} x \\ y \\ \theta \\ v_x \\ v_y \\ \omega \end{bmatrix}_{k-1} + \begin{bmatrix} v_x\,\mathrm{cs} - v_y\,\mathrm{sn} \\ v_x\,\mathrm{sn} + v_y\,\mathrm{cs} \\ T_s\omega \\ T_s v_y\omega + T_s\lambda_a u_1 \\ -T_s v_x\omega + T_s\lambda_a u_2 \\ T_s\lambda_\alpha\gamma u_3 \end{bmatrix}_{k-1}, \begin{array}{l}\mathrm{cs} = T_s\cos\theta \\ \mathrm{sn} = T_s\sin\theta\end{array}
\end{aligned} \tag{3.10}$$

$$\Rightarrow A_{k-1} = \left.\frac{\partial f}{\partial x}\right|_{\substack{x_{k-1}\\u_{k-1}}} = \begin{bmatrix} 1 & 0 & -v_x\,\mathrm{sn} - v_y\,\mathrm{cs} & \mathrm{cs} & -\mathrm{sn} & 0 \\ 0 & 1 & v_x\,\mathrm{cs} - v_y\,\mathrm{sn} & \mathrm{sn} & \mathrm{cs} & 0 \\ 0 & 0 & 1 & 0 & 0 & T_s \\ 0 & 0 & 0 & 1 & T_s\omega & T_s v_y \\ 0 & 0 & 0 & -T_s\omega & 1 & -T_s v_x \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{\substack{x_{k-1}\\u_{k-1}}}$$

Para la etapa de predicción, se separa A_k en 4 componentes: A_p con el modelo de la *pose*, A_{pv} con la influencia de la velocidad en la postura, A_{vp} con la influencia de la postura en la velocidad (nula en el modelo linealizado (3.10)) y A_v con el modelo de la velocidad; tal y como se muestra en la ecuación (3.11).

$$A_{k-1} = \left.\frac{\partial f}{\partial x}\right|_{x_{k-1},u_{k-1}} = \begin{bmatrix} A_p & A_{pv} \\ A_{vp} & A_v \end{bmatrix}_{k-1}, A_p = \begin{bmatrix} 1 & 0 & -v_x\,\mathrm{sn} - v_y\,\mathrm{cs} \\ 0 & 1 & v_x\,\mathrm{cs} - v_y\,\mathrm{sn} \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

Se definen y seccionan las matrices de covarianza del error P_k y covarianza del ruido del proceso Q_k para el modelo (B.51), según se muestran en las ecuaciones (3.12) y (3.13) respectivamente.

$$P_{k-1} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} & P_{15} & P_{16} \\ P_{21} & P_{22} & P_{23} & P_{24} & P_{25} & P_{26} \\ P_{31} & P_{32} & P_{33} & P_{34} & P_{35} & P_{36} \\ P_{41} & P_{42} & P_{43} & P_{44} & P_{45} & P_{46} \\ P_{51} & P_{52} & P_{53} & P_{54} & P_{55} & P_{56} \\ P_{61} & P_{62} & P_{63} & P_{64} & P_{65} & P_{66} \end{bmatrix}_{k-1} = \begin{bmatrix} P_p & P_{pv} \\ P_{vp} & P_v \end{bmatrix}_{k-1} \tag{3.12}$$

$$Q_{k-1} = \begin{bmatrix} Q_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & Q_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & Q_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & Q_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & Q_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & Q_{66} \end{bmatrix}_{k-1} = \begin{bmatrix} Q_p & Q_{pv} \\ Q_{vp} & Q_v \end{bmatrix}_{k-1} \tag{3.13}$$

El valor exacto de la matriz Q_k puede determinarse de forma precisa para sistemas lineales a partir de la matriz de covarianza del proceso continua Qc (la cual pocas veces es conocida de forma exacta), aunque puede resultar un proceso complejo en el caso de sistemas no lineales de dimensiones altas. Esto debido a que se requiere el cálculo de la integral $Q_{k-1} = \int_{t_{k-1}}^{t_k} e^{A(t_k-\tau)} Q_c(\tau)\, e^{A^T(t_k-\tau)} d\tau$ tal y como describen los trabajos [30, 72, 198, 166], por lo que en estos casos se suele determinar mediante distintas aproximaciones. El enfoque seguido en la presente tesis se basa en utilizar una matriz Q_k diagonal y ajustarla de acuerdo con la plataforma experimental en la que se desea implementar. Este procedimiento es común aun en los casos donde se conoce su valor exacto, tal y como se defiende en [30, 72, 198, 166].

Siguiendo la definición del algoritmo 3, la predicción de la covarianza del error P_k^- se realiza utilizando las matrices A_k, P_k y Q_k definidas en las ecuaciones (3.11), (3.12) y (3.13) junto con W_k que es igual a la matriz identidad 6×6, tal y como se muestra en la ecuación (3.14) (donde por simplicidad se ha omitido el subíndice $k-1$).

$$\begin{aligned}
P_k &= A_{k-1} P_{k-1} A_{k-1}^T + Q_{k-1} \\
&= \begin{bmatrix} A_p & A_{pv} \\ A_{vp} & A_v \end{bmatrix} \begin{bmatrix} P_p & P_{pv} \\ P_{vp} & P_v \end{bmatrix} \begin{bmatrix} A_p & A_{pv} \\ A_{vp} & A_v \end{bmatrix}^T + \begin{bmatrix} Q_p & Q_{pv} \\ Q_{vp} & Q_v \end{bmatrix} \\
&= \begin{bmatrix} A_p & A_{pv} \\ A_{vp} & A_v \end{bmatrix} \begin{bmatrix} P_p & P_{pv} \\ P_{vp} & P_v \end{bmatrix} \begin{bmatrix} A_p^T & A_{vp}^T \\ A_{pv}^T & A_v^T \end{bmatrix} + \begin{bmatrix} Q_p & Q_{pv} \\ Q_{vp} & Q_v \end{bmatrix} \\
&= \begin{bmatrix} A_p P_p + A_{pv} P_{vp} & A_p P_{pv} + A_{pv} P_v \\ A_{vp} P_p + A_v P_{vp} & A_{vp} P_{pv} + A_v P_v \end{bmatrix} \begin{bmatrix} A_p^T & A_{vp}^T \\ A_{pv}^T & A_v^T \end{bmatrix} + \begin{bmatrix} Q_p & Q_{pv} \\ Q_{vp} & Q_v \end{bmatrix} \\
\because \quad & \Omega_p = (A_p P_p + A_{pv} P_{vp}), \Omega_{pv} = (A_p P_{pv} + A_{pv} P_v), \\
& \Omega_{vp} = (A_{vp} P_p + A_v P_{vp}), \Omega_v = (A_{vp} P_{pv} + A_v P_v) \\
\Rightarrow P_k &= \begin{bmatrix} \Omega_p & \Omega_{pv} \\ \Omega_{vp} & \Omega_v \end{bmatrix} \begin{bmatrix} A_p^T & A_{vp}^T \\ A_{pv}^T & A_v^T \end{bmatrix} + \begin{bmatrix} Q_p & Q_{pv} \\ Q_{vp} & Q_v \end{bmatrix} \\
\therefore P_k &= \begin{bmatrix} \Omega_p A_p^T + \Omega_{pv} A_{pv}^T + Q_p & \Omega_p A_{vp}^T + \Omega_{pv} A_v^T + Q_{pv} \\ \Omega_{vp} A_p^T + \Omega_v A_{pv}^T + Q_{vp} & \Omega_{vp} A_{vp}^T + \Omega_v A_v^T + Q_v \end{bmatrix}
\end{aligned} \tag{3.14}$$

Se observa en las ecuaciones (3.10) y (3.11) que la matriz A_{vp} es nula de dimensiones $0_{3\times3}$ y que además los términos Q_{vp} y Q_{pv} también lo son cuando Q_k es una matriz

diagonal como en (3.13). De este modo es posible simplificar la ecuación (3.14) en (3.15).

$$\begin{aligned} &\because A_{vp} = 0_{3\times 3} \\ &\Rightarrow \Omega_p = \left(A_p P_p + A_{pv} P_{vp}\right), \Omega_{pv} = \left(A_p P_{pv} + A_{pv} P_v\right), \\ &\Omega_{vp} = \left(A_v P_{vp}\right), \Omega_v = \left(A_v P_v\right) \\ &\therefore P_k = \begin{bmatrix} \Omega_p A_p^T + \Omega_{pv} A_{pv}^T + Q_p & \Omega_{pv} A_v^T + Q_{pv} \\ \Omega_{vp} A_p^T + \Omega_v A_{pv}^T + Q_{vp} & \Omega_v A_v^T + Q_v \end{bmatrix} = \begin{bmatrix} P_p & P_{pv} \\ P_{vp} & P_v \end{bmatrix}_k \end{aligned} \tag{3.15}$$

Una vez definida la etapa de predicción, es necesario realizar la corrección, en primer lugar, la corrección local basada en velocidades. Para ello se define la matriz de medición H_k y la covarianza del ruido en la medición R_k, tal y como muestran las ecuaciones (3.16) y (3.17) respectivamente.

$$z_k = H_k x_k \Rightarrow H_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}_k = \begin{bmatrix} H_p & H_{pv} \\ H_{vp} & H_v \end{bmatrix}_k \tag{3.16}$$

$$R_k = \begin{bmatrix} R_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & R_{22} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & R_{33} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & R_{44} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & R_{55} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & R_{66} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & R_{77} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_{88} \end{bmatrix}_k = \begin{bmatrix} R_p & R_{pv} \\ R_{vp} & R_v \end{bmatrix}_k \tag{3.17}$$

Este planteamiento permite reducir el coste computacional, al calcular una matriz inversa 5×5 para el filtro local (solo utiliza M_v^{-1}) y una inversa de 3×3 para el filtro global (solo requiere M_p^{-1}) en lugar de una matriz 8×8.

La etapa de corrección global resultará igual que para el caso del filtro completo, con la diferencia de que el filtro completo actualizaba los términos locales y globales al mismo tiempo y para el filtro en cascada estas actualizaciones se realizan primero para los términos locales y posteriormente para los globales.

Así pues, el esquema matemático del filtro en cascada queda como muestra la ecuación (3.18). Su implementación representada en formato de pseudocódigo se muestra en el algoritmo 5.

EKF en Cascada, modelo (B.51)

$\because A_{vp} = 0_{3\times3}, H_{vp} = R_{vp} = 0_{5\times3}, H_{pv} = 0_{3\times3}, R_{pv} = 0_{3\times5}$

Predicción Local:

$$\hat{x}_k = f\left(\hat{x}_{k-1}, u_{k-1}, 0\right)$$

$$A_{k-1} = \left.\frac{\partial f}{\partial x}\right|_{x_{k-1},u_{k-1}} = \begin{bmatrix} A_p & A_{pv} \\ A_{vp} & A_v \end{bmatrix}, \Omega_p = (A_p P_p + A_{pv} P_{vp}), \Omega_v = (A_v P_v), \Omega_{pv} = (A_p P_{pv} + A_{pv} P_v), \Omega_{vp} = (A_v P_{vp})$$

$$P_k = A_{k-1} P_{k-1} A_{k-1}^T + Q_{k-1} = \begin{bmatrix} \Omega_p A_p^T + \Omega_{pv} A_{pv}^T + Q_p & \Omega_{pv} A_v^T + Q_{pv} \\ \Omega_{vp} A_p^T + \Omega_v A_{pv}^T + Q_{vp} & \Omega_v A_v^T + Q_v \end{bmatrix}$$

Corrección Local: $\because H_p = 0_{3\times3}$

$$M_v^{-1} = \left(H_v P_v H_v^T + R_v\right)^{-1} \tag{3.18}$$

$$K_{k,v} = \begin{bmatrix} 0 & P_{pv} H_v^T N_v \\ 0 & P_v H_v^T N_v \end{bmatrix} = \begin{bmatrix} 0 & P_{pv} H_v^T M_v^{-1} \\ 0 & P_v H_v^T M_v^{-1} \end{bmatrix} = \begin{bmatrix} 0 & K_{pv} \\ 0 & K_v \end{bmatrix}$$

$$\hat{x}_k = \hat{x}_k + K_k\left(z_k - H_k \hat{x}_k\right) = \begin{bmatrix} \hat{x}_{k,p} \\ \hat{x}_{k,v} \end{bmatrix} + \begin{bmatrix} K_{pv}\Delta_{k,v} \\ K_v \Delta_{k,v} \end{bmatrix}, \Delta_{k,v} = z_{k,v} - H_v \hat{x}_{k,v}$$

$$P_k = P_k - K_k H P_k = \begin{bmatrix} P_p - K_{pv} H_v P_{vp} & P_{pv} - K_{pv} H_v P_v \\ P_{vp} - K_v H_v P_{vp} & P_v - K_v H_v P_v \end{bmatrix}$$

Corrección Global: $\because H_v = 0_{5\times3}, K_{vp} = 0_{3\times3}$

$$K_{k,p} = P_p H_p^T N_p = P_p H_p^T M_p^{-1} = P_p H_p^T \left(H_p P_p H_p^T + R_p\right)^{-1}$$

$$\hat{x}_{k,p} = \hat{x}_{k,p} + K_{k,p}\left(z_{k,p} - H_{k,p}\hat{x}_{k,p}\right)$$

$$P_{k,p} = P_{k,p} - K_{k,p} H_{k,p} P_{k,p}$$

Algoritmo 5: Algoritmo EKF recursivo en cascada, asignación de entradas, predicción local y corrección global continua

Entrada: $u_{k-1} = [\ u_1 \quad u_2 \quad u_3\ ]^T$,$\hat{x}_{k-1}$,$P_{k-1}$

Medición: $z_k = [\ z_{k,p} \quad z_{k,v}\ ]^T$, $z_{k,v} = [\ v_{x,enc}\ v_{y,enc}\ \omega_{enc}\ \omega_{gyr}\ \omega_{comp}\ ]_k^T$
$z_{k,p} = L_{GP} = [\ x_k\ y_k\ \theta_k\ ]_{GP}^T$

Datos: $f(\cdot)$ *y* $h(\cdot)$ *(del modelo no lineal* (B.24) *o* (B.51)*)*,Q_k,
$R_k = [R_p\ R_{pv}; R_{vp}\ R_v]_k$

Salida: $\hat{x}_k = [\ x_{k,p} \quad x_{k,v}\ ]^T$,$P_k = [P_p\ P_{pv}; P_{vp}\ P_v]_k$

Inicialización: $\hat{x}_0$,P_0

Para *el instante actual* k **hacer**

Predicción:

$\hat{x}_k = f(\hat{x}_{k-1}, u_{k-1}, 0)$

$A_k = \frac{\partial f}{\partial x}\Big|_{\hat{x}_{k-1},u_{k-1},0}$, $W_k = \frac{\partial f}{\partial w}\Big|_{\hat{x}_{k-1},u_{k-1},0}$

$P_k = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$

$H_k = \frac{\partial h}{\partial x}\Big|_{\hat{x}_k,0} = [H_p\ H_{pv}; H_{vp}\ H_v]_k = [H_p\ 0; 0\ H_v]_k$, $N_k = \frac{\partial h}{\partial n}\Big|_{\hat{x}_k,0}$

Corrección Local: utilizar H_p nula para actualizar con $z_{k,v}$

$K_k = P_k H_k^T \left(H_k P_k H_k^T + N_k R_k N_k^T\right)^{-1}$

$\hat{x}_k = \hat{x}_k + K_k [z_k - h(\hat{x}_k, 0)]$

$P_k = (I - K_k H_k) P_k$

Corrección Global: actualizar únicamente $(\hat{x}_{k,p}, P_{k,p})$ con $z_{k,p}$

$K_{k,p} = P_{k,p} H_{k,p}^T \Big(H_{k,p} P_{k,p} H_{k,p}^T + R_{k,p}\Big)^{-1}$

$\hat{x}_{k,p} = \hat{x}_{k,p} + K_{k,p}(z_{k,p} - H_{k,p}\hat{x}_{k,p})$

$P_{k,p} = P_{k,p} - K_{k,p} H_{k,p} P_{k,p}$

fin

Llegados a este punto, cabe destacar que la versión del EKF del algoritmo 5 es el óptimo conseguido en este trabajo en cuanto a coste computacional para sistemas no lineales. Más adelante se exponen los resultados empíricos cuantitativos que lo justifican. A continuación se expone el mismo procedimiento pero aplicado a los modelos lineales.

3.2.1.2 Reducción del Coste Computacional de los Algoritmos de Fusión Basados en Modelos Dinámicos Lineales

Partiendo de los modelos dinámicos lineales de la configuración diferencial (ecuación (B.23)) y Ackerman (ecuación (B.52)) y del desarrollo del filtro en cascada expuesto previamente, se secciona el estado x_k según los estados locales $x_{k,v}$ (velocidad) y globales $x_{k,p}$ (*pose*) de la misma forma que se secciona la medición en sensores de velocidad $z_{k,v}$ y la postura global $z_{k,p}$ según se muestra en la ecuación (3.19) para el caso con deslizamiento y en la (3.20) sin deslizamiento. En estas ecuaciones se ha definido además los vectores de entrada $u_{k,Aks}$ para la plataforma Ackerman con deslizamiento, $u_{k,Akns}$ del Ackerman sin deslizamiento y $u_{k,dif}$ para el robot diferencial sin deslizamiento.

Con ésta asignación, el procedimiento a seguir es emplear el filtro local para realizar la fusión sensorial utilizando u_k, $x_{k,v}$ y $z_{k,v}$ junto con los modelos locales de velocidad (ecuaciones (B.50), (B.52) para el robot Ackerman y (B.23) para el robot diferencial). A continuación utilizar la salida del filtro local como entrada del modelo cinemático global (ecuaciones (B.11) y (B.36) para el robot diferencial y Ackerman respectivamente) para obtener la postura global. Y finalmente actualizar la postura utilizando $z_{k,p}$ y la corrección global del filtro en cascada reducido de la ecuación (3.18).

La principal ventaja de este planteamiento es el número reducido de operaciones a efectuar comparado con el filtro EKF de la ecuación (3.18). Además para este caso, es posible utilizar un filtro KF lineal para la fusión local dado que se utilizan los modelos lineales sin deslizamiento de las ecuaciones (B.23) y (B.52).

$$\begin{aligned} \boldsymbol{x}_{k,p} &= \begin{bmatrix} x & y & \theta \end{bmatrix}_k^T, \quad \boldsymbol{x}_{k,v} = \begin{bmatrix} v_x & v_y & \omega \end{bmatrix}_k^T \\ \boldsymbol{u}_{k,Aks} &= \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}_k^T \\ \boldsymbol{z}_{k,p} &= H_{k,p} x_{k,p} = \boldsymbol{L}_{GP} = \begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}_{GP}^T \\ \boldsymbol{z}_{k,v} &= H_{k,v} x_{k,v} = \begin{bmatrix} v_{x,enc} & v_{y,enc} & \omega_{enc} & \omega_{gyr} & \omega_{comp} \end{bmatrix}_k^T \end{aligned} \tag{3.19}$$

$$\begin{aligned} \boldsymbol{x}_{k,p} &= \begin{bmatrix} x & y & \theta \end{bmatrix}_k^T, \quad \boldsymbol{x}_{k,v} = \begin{bmatrix} v_x & \omega \end{bmatrix}_k^T \\ \boldsymbol{u}_{k,Akns} &= \begin{bmatrix} u_1 & u_3 \end{bmatrix}_k^T, \boldsymbol{u}_{k,dif} = \begin{bmatrix} a & \alpha \end{bmatrix}_k^T \\ \boldsymbol{z}_{k,p} &= H_{k,p} x_{k,p} = \boldsymbol{L}_{GP} = \begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}_{GP}^T \\ \boldsymbol{z}_{k,v} &= H_{k,v} x_{k,v} = \begin{bmatrix} v_{x,enc} & \omega_{enc} & \omega_{gyr} & \omega_{comp} \end{bmatrix}_k^T \end{aligned} \tag{3.20}$$

El planteamiento de este filtro se puede realizar a partir del filtro en cascada EKF (3.18) al considerar las mismas definiciones para Q_k (ecuación (3.13)), H_k (ecuación (3.16)) y R_k (ecuación (3.17)) según se describe a continuación para el

caso con deslizamiento de la ecuación (3.19). El filtro local de velocidad se obtiene al sustituir $A_p = A_{pv} = A_{vp} = 0_{3\times3}$ en las etapas de predicción y corrección local y al considerar un único término Q_v según se muestra en la ecuación (3.21). Esto debido a que solamente se utiliza el modelo local no lineal (ecuación (B.50)) para realizar la fusión de velocidad con lo que solo se requiere el cálculo de A_v (cuyo valor es el mismo al obtenido en las ecuaciones (3.10) y (3.11)).

$$
\begin{aligned}
&\because A_{vp}=A_{pv}=A_p=0_{3\times3}, H_{vp}=R_{vp}=0_{5\times3}, H_{pv}=0_{3\times3}, R_{pv}=0_{3\times5}\\
&\text{Predicción Local:}\\
&\hat{x}_k = f\left(\hat{x}_{k-1}, u_{k-1}, 0\right)\\
&A_{k-1}=\left.\frac{\partial f}{\partial x}\right|_{x_{k-1},u_{k-1}}=\begin{bmatrix}0 & 0\\ 0 & A_v\end{bmatrix}, A_v=\left[\begin{array}{ccc}1 & T_s\omega & T_s v_y\\ -T_s\omega & 1 & -T_s v_x\\ 0 & 0 & 1\end{array}\right]_{\substack{x_{k-1}\\ u_{k-1}}}\\
&\quad \Omega_p=0 \quad \Omega_v=(A_v P_v) \quad \Omega_{pv}=0 \quad \Omega_{vp}=(A_v P_{vp})\\
&P_k=A_{k-1}P_{k-1}A_{k-1}^T+Q_{k-1}=\begin{bmatrix}0 & 0\\ 0 & \Omega_v A_v^T+Q_v\end{bmatrix}=\begin{bmatrix}P_p & P_{pv}\\ P_{vp} & P_v\end{bmatrix}\\
&\text{Corrección Local: } \because H_p = 0_{3\times3}\\
&\quad M_v^{-1} = \left(H_v P_v H_v^T + R_v\right)^{-1}\\
&K_{k,v}=\begin{bmatrix}0 & P_{pv}H_v^T M_v^{-1}\\ 0 & P_v H_v^T M_v^{-1}\end{bmatrix}=\begin{bmatrix}0 & 0\\ 0 & K_v\end{bmatrix}\\
&\hat{x}_{k,v}=\hat{x}_{k,v}+K_k\left(z_{k,v}-H_{k,v}\hat{x}_{k,v}\right)\\
&P_{k,v} = P_{k,v} - K_{k,v}H_{k,v}P_{k,v}
\end{aligned}
\tag{3.21}
$$

Con la estimación de la velocidad $x_{k,v}$ de la ecuación (3.21) en la que se han incorporado las mediciones de los sensores locales, se procede a utilizar el modelo cinemático global (B.36) para obtener la postura $\hat{x}_{k,p} = f_p\left(\hat{x}_{k,p}, \hat{x}_{k,v}\right)$ al utilizar $x_{k,v}$ como entrada, según se muestra en la ecuación (3.22).

$$
\begin{aligned}
x_{k,p} &= \begin{bmatrix} x & y & \theta \end{bmatrix}_k^T, \quad x_{k,v} = \begin{bmatrix} v_x & v_y & \omega \end{bmatrix}_k^T\\
\hat{x}_{k,p} &= \hat{x}_{k-1,p} + T_s \begin{bmatrix}\cos\theta_{k-1} & -\sin\theta_{k-1} & 0\\ \sin\theta_{k-1} & \cos\theta_{k-1} & 0\\ 0 & 0 & 1\end{bmatrix} \hat{x}_{k,v} = f_p\left(\hat{x}_{k-1,p}, \hat{x}_{k,v}\right)
\end{aligned}
\tag{3.22}
$$

Para actualizar la postura utilizando la información global $z_{k,p}$ se utiliza la etapa de corrección global del filtro en cascada reducido (3.18), la cual requiere el valor de $P_{k,p}$ para el cálculo de la ganancia del filtro. Sin embargo, a diferencia del filtro cascada reducido, el término $P_{k,p}$ no es calculado por el filtro local de la ecuación (3.21) que solo obtiene $P_{k,v}$.

Para determinar $P_{k,p}$ se propone utilizar su valor en el instante anterior junto con la propagación de la covarianza $P_{k,v}$ a través del modelo cinemático global que relaciona la velocidad con la *pose*. Para esto primeramente se linealiza el modelo global respecto a $(\hat{x}_{k,p}, \hat{x}_{k,v})$ tal y como se muestra en la ecuación (3.23), con lo que se obtiene el mismo resultado que para el A_p y A_{pv} del filtro en cascada resumido (ecuación (3.21)) al ser el modelo dinámico global (B.51) deducido mediante el cinemático global (B.36).

$$\begin{aligned} \mathrm{cs} = T_s \cos\theta, \mathrm{sn} = T_s \sin\theta \\ A_{k,p} = \left.\frac{\partial f}{\partial x_p}\right|_{\substack{x_{k-1} \\ u_{k-1}}} = \left[\begin{array}{ccc} 1 & 0 & -v_x\,\mathrm{sn} - v_y\,\mathrm{cs} \\ 0 & 1 & v_x\,\mathrm{cs} - v_y\,\mathrm{sn} \\ 0 & 0 & 1 \end{array}\right]_{\substack{x_{k-1} \\ u_{k-1}}} \\ A_{k,pv} = \left.\frac{\partial f}{\partial x_v}\right|_{\substack{x_{k-1} \\ u_{k-1}}} = \left[\begin{array}{ccc} \mathrm{cs} & -\mathrm{sn} & 0 \\ \mathrm{sn} & \mathrm{cs} & 0 \\ 0 & 0 & T_s \end{array}\right]_{\substack{x_{k-1} \\ u_{k-1}}} \end{aligned} \tag{3.23}$$

A partir de la ecuación de predicción de la covarianza del error P_k del filtro en cascada (ecuación (3.18)) se observa que la covarianza del error de estimación de la *pose* P_p se puede determinar mediante la ecuación (3.24).

$$\begin{aligned} \Omega_p = \left(A_p P_p + A_{pv} P_{vp}\right), \Omega_{pv} = \left(A_p P_{pv} + A_{pv} P_v\right), \\ P_p = \Omega_p A_p^T + \Omega_{pv} A_{pv}^T + Q_p \\ \Rightarrow P_p = \left(A_p P_p + A_{pv} P_{vp}\right) A_p^T + \left(A_p P_{pv} + A_{pv} P_v\right) A_{pv}^T + Q_p \\ \Rightarrow P_p = A_p P_p A_p^T + A_{pv} P_{vp} A_p^T + A_p P_{pv} A_{pv}^T + A_{pv} P_v A_{pv}^T + Q_p \end{aligned} \tag{3.24}$$

El filtro local de la ecuación (3.21) no realiza la estimación de los términos (P_{pv}, P_{vp}), los cuales no se requieren en el cálculo de $x_{k,p}$ realizado en la ecuación (3.22). De esta forma, al no disponer de estos términos para el cálculo de P_p se debe incorporar su influencia mediante el término Q_p, incrementando su magnitud para obtener un valor de P_p cercano al obtenido mediante la ecuación (3.24). De esta forma al eliminar los términos (P_{pv}, P_{vp}) se simplifica la ecuación (3.24) en (3.25) con la que se obtiene la P_p requerida. Se observa que P_p depende de la propagación de la covarianza del instante anterior al actual mediante A_p, de la propagación de la covarianza asociada a la velocidad P_v mediante A_{pv} y la matriz de covarianza del ruido del proceso (postura) Q_p. Como aproximación adicional se puede omitir la propagación de $P_{k-1,p}$ en A_p, al considerar que ésta se aproxima mediante el valor de $P_{k-1,p}$, evitando el cálculo de A_p con lo que se requieren menos operaciones

para implementar el filtro. Esta aproximación supone un mayor aporte de P_v en P_p y su precisión puede mejorarse con el respectivo ajuste de valor de Q_p.

$$\begin{aligned} P_{k,p} &= A_{k,p}P_{k-1,p}A_{k,p}^T + A_{k,pv}P_{k,v}A_{k,pv}^T + Q_{k,p} \\ P_{k,p} &\approx P_{k-1,p} + A_{k,pv}P_{k,v}A_{k,pv}^T + Q_{k,p} \end{aligned} \tag{3.25}$$

Obtenido el valor de P_p se procede a actualizar la *pose* utilizando la información global $z_{k,p}$ mediante la etapa de corrección global del filtro en cascada reducido (3.18), tal y como se muestra en la ecuación (3.26), en donde H_p es la matriz identidad 3×3 (según su definición en las ecuaciones (3.16) y (3.19)).

$$\begin{aligned} K_{k,p} &= P_p H_p^T \left(H_p P_p H_p^T + R_p\right)^{-1} \\ \hat{x}_{k,p} &= \hat{x}_{k,p} + K_{k,p}\left(z_{k,p} - H_{k,p}\hat{x}_{k,p}\right) \\ P_{k,p} &= P_{k,p} - K_{k,p}H_{k,p}P_{k,p} \end{aligned} \tag{3.26}$$

Con este resultado se ha obtenido la *pose* actualizada mediante las etapas de predicción y corrección local (velocidad, ecuación (3.21)), modelo de la postura en cascada (salida del modelo de velocidad utilizado como entrada al modelo de posición) y su covarianza del error (ecuaciones (3.22), (3.23) y (3.25)) y actualización global (ecuación (3.26)), procedimiento que se resume en la ecuación (3.27) y cuyo algoritmo en pseudocódigo se muestra en el Algoritmo 6.

Cabe destacar que al utilizar el filtro de fusión de la ecuación (3.27) con los modelos local y global en cascada, se está considerando una aproximación al problema con el fin de reducir el número de operaciones requeridas. No se utilizan las covarianzas cruzadas entre el proceso local (velocidades) y el global (postura) para determinar x_k y P_k.

EKF en Cascada con Modelo local (B.51)/**global**(B.36) **en cascada**

Predicción Local:

$$\hat{x}_{k,v} = f_v\left(\hat{x}_{k-1,v}, u_{k-1}, 0\right)$$

$$A_{k-1,v} = \left.\frac{\partial f_v}{\partial x}\right|_{x_{k-1},u_{k-1}} = \left[\begin{array}{ccc} 1 & T_s\omega & T_s v_y \\ -T_s\omega & 1 & -T_s v_x \\ 0 & 0 & 1 \end{array}\right]_{\substack{x_{k-1} \\ u_{k-1}}}$$

$$P_{k,v} = A_{k-1,v} P_{k-1,v} A_{k-1,v}^T + Q_{k-1,v}$$

Corrección Local:

$$K_{k,v} = P_{k,v} H_{k,v}^T \left(H_{k,v} P_{k,v} H_{k,v}^T + R_{k,v}\right)^{-1}$$

$$\hat{x}_{k,v} = \hat{x}_{k,v} + K_{k,v}\left(z_{k,v} - H_{k,v}\hat{x}_{k,v}\right)$$

$$P_{k,v} = P_{k,v} - K_{k,v} H_{k,v} P_{k,v}$$

Modelo Global, obtener postura $\hat{x}_{k,p} = f_p\left(\hat{x}_{k,p}, \hat{x}_{k,v}\right)$:

$$\hat{x}_{k,p} = \hat{x}_{k-1,p} + T_s \left[\begin{array}{ccc} \cos\theta_{k-1} & -\sin\theta_{k-1} & 0 \\ \sin\theta_{k-1} & \cos\theta_{k-1} & 0 \\ 0 & 0 & 1 \end{array}\right] \hat{x}_{k,v} = f_p\left(\hat{x}_{k-1,p}, \hat{x}_{k,v}\right)$$

Covarianza P_p: propagar P_v a través del modelo global:

$\mathrm{cs} = T_s \cos\theta, \mathrm{sn} = T_s \sin\theta$

$$A_{k,p} = \left.\frac{\partial f}{\partial x_p}\right|_{\substack{x_{k-1} \\ u_{k-1}}} = \left[\begin{array}{ccc} 1 & 0 & -v_x\,\mathrm{sn} - v_y\,\mathrm{cs} \\ 0 & 1 & v_x\,\mathrm{cs} - v_y\,\mathrm{sn} \\ 0 & 0 & 1 \end{array}\right]_{\substack{x_{k-1} \\ u_{k-1}}}, A_{k,pv} = \left.\frac{\partial f}{\partial x_v}\right|_{\substack{x_{k-1} \\ u_{k-1}}} = \left[\begin{array}{ccc} \mathrm{cs} & -\mathrm{sn} & 0 \\ \mathrm{sn} & \mathrm{cs} & 0 \\ 0 & 0 & T_s \end{array}\right]_{\substack{x_{k-1} \\ u_{k-1}}}$$

$$P_{k,p} = A_{k,p} P_{k-1,p} A_{k,p}^T + A_{k,pv} P_{k,v} A_{k,pv}^T + Q_{k,p} \approx P_{k-1,p} + A_{k,pv} P_{k,v} A_{k,pv}^T + Q_{k,p}$$

Corrección Global:

$$K_{k,p} = P_p H_p^T \left(H_p P_p H_p^T + R_p\right)^{-1}$$

$$\hat{x}_{k,p} = \hat{x}_{k,p} + K_{k,p}\left(z_{k,p} - H_{k,p}\hat{x}_{k,p}\right)$$

$$P_{k,p} = P_{k,p} - K_{k,p} H_{k,p} P_{k,p}$$

(3.27)

Algoritmo 6: Algoritmo EKF recursivo en cascada con modelos local/global en cascada, asignación de entradas, predicción local de velocidades, modelo global de *pose* y corrección global continua

Entrada: $u_{k-1} = u_{Aks} = [\, u_1 \;\; u_2 \;\; u_3 \,]^T,\ \hat{x}_{k-1,v}, P_{k-1,v},\ \hat{x}_{k-1,p}, P_{k-1,p}$

Medición: $z_{k,v} = [\; v_{x,enc} \;\; v_{y,enc} \;\; \omega_{enc} \;\; \omega_{gyr} \;\; \omega_{comp} \;\;]^T_k$,
$z_{k,p} = [\; x_k \;\; y_k \;\; \theta_k \;\;]^T_{GP}$

Datos: $f_v(\cdot)$ *y* $h_v(\cdot)$ *del modelo no lineal local* (B.50),$Q_{k,v}$,$R_{k,v}$,
$f_p(\cdot)$ *y* $h_p(\cdot)$ *del modelo no lineal global* (B.36),$Q_{k,p}$,$R_{k,p}$

Salida: $\hat{x}_{k,v} = \left[\; v_x \quad v_y \quad \omega \;\right]^T_k$,$\hat{x}_{k,p} = \left[\; x \quad y \quad \theta \;\right]^T_k$,$P_{k,p},\ P_{k,v}$

Inicialización: $\hat{x}_{0,v}$,$P_{0,v}$,$\hat{x}_{0,p}$,$P_{0,p}$

Para *el instante actual* k **hacer**

Predicción Local:

$$\hat{x}_{k,v} = f_v\left(\hat{x}_{k-1,v}, u_{k-1}, 0\right)$$

$$A_{k-1,v} = \left.\frac{\partial f_v}{\partial x_v}\right|_{\substack{x_{k-1} \\ u_{k-1}}},\ W_{k-1,v} = \left.\frac{\partial f_v}{\partial w_v}\right|_{\hat{x}_{k-1,v}, u_{k-1}, 0}$$

$$P_{k,v} = A_{k-1,v} P_{k-1,v} A^T_{k-1,v} + W_{k-1,v} Q_{k-1,v} W^T_{k-1,v}$$

$$H_{k,v} = \left.\frac{\partial h_v}{\partial x_v}\right|_{\hat{x}_k, 0},\ N_{k,v} = \left.\frac{\partial h_v}{\partial n_v}\right|_{\hat{x}_k, 0}$$

Corrección Local:

$$K_{k,v} = P_{k,v} H^T_{k,v} \left(H_{k,v} P_{k,v} H^T_{k,v} + N_{k,v} R_{k,v} N^T_{k,v} \right)^{-1}$$

$$\hat{x}_{k,v} = \hat{x}_{k,v} + K_{k,v}\left[z_{k,v} - h_{k,v}\left(\hat{x}_{k,v}, 0\right) \right]$$

$$P_{k,v} = \left(I - K_{k,v} H_{k,v}\right) P_{k,v}$$

Modelo Global, Covarianza Global:

$$\hat{x}_{k,p} = f_p\left(\hat{x}_{k-1,p}, \hat{x}_{k,v}\right)$$

$$A_{k,p} = \left.\frac{\partial f_p}{\partial x_p}\right|_{\substack{x_{k-1} \\ u_{k-1}}},\ A_{k,pv} = \left.\frac{\partial f_p}{\partial x_v}\right|_{\substack{x_{k-1} \\ u_{k-1}}}$$

$$P_{k,p} = A_{k,p} P_{k-1,p} A^T_{k,p} + A_{k,pv} P_{k,v} A^T_{k,pv} + Q_{k,p}$$

Corrección Global: $H_{k,p} = \left.\frac{\partial h_p}{\partial x_p}\right|_{\hat{x}_k, 0}$ =Identidad 3×3 según ecuación (3.19)

$$K_{k,p} = P_{k,p} H^T_{k,p} \left(H_{k,p} P_{k,p} H^T_{k,p} + R_{k,p} \right)^{-1}$$

$$\hat{x}_{k,p} = \hat{x}_{k,p} + K_{k,p}\left(z_{k,p} - H_{k,p} \hat{x}_{k,p} \right)$$

$$P_{k,p} = P_{k,p} - K_{k,p} H_{k,p} P_{k,p}$$

fin

Como se puede observar, el algoritmo requiere el cálculo de dos matrices inversas de las mismas dimensiones que el filtro en cascada (3.18), por lo que para casos de modelos de velocidad con deslizamiento (no lineales) su aplicación no está del todo justificada. Sin embargo, como ya se ha comentado anteriormente, para el caso de los modelos de velocidad sin deslizamiento ((B.23) para el robot diferencial y (B.52) para el robot Ackerman) se consigue la ventaja adicional de poder utilizar un KF como filtro local. De este modo se obtiene el esquema del procedimiento (3.28) y su implementación en el algoritmo 7.

KF en Cascada con Modelos local/global en cascada

Predicción Local:

$$\hat{x}_{k,v} = A_{k-1,v}\hat{x}_{k-1,v} + B_{k-1,v}u_{k-1}$$

$$P_{k,v}=A_{k-1,v}P_{k-1,v}A_{k-1,v}^T+Q_{k-1,v}$$

Corrección Local:

$$K_{k,v}= P_{k,v}H_{k,v}^T\left(H_{k,v}P_{k,v}H_{k,v}^T + R_{k,v}\right)^{-1}$$

$$\hat{x}_{k,v}=\hat{x}_{k,v}+K_{k,v}\left(z_{k,v}-H_{k,v}\hat{x}_{k,v}\right)$$

$$P_{k,v} = P_{k,v} - K_{k,v}H_{k,v}P_{k,v}$$

Modelo Global, obtener postura $\hat{x}_{k,p}$:

$$\hat{x}_{k,p}= f_p\left(\hat{x}_{k-1,p}, \hat{x}_{k,v}\right) \tag{3.28}$$

Covarianza P_p: propagar P_v a través del modelo global:

$$A_{k,p}=\left.\frac{\partial f}{\partial x_p}\right|_{\substack{x_{k-1}\\u_{k-1}}}, A_{k,pv}=\left.\frac{\partial f}{\partial x_v}\right|_{\substack{x_{k-1}\\u_{k-1}}}$$

$$\begin{aligned}P_{k,p}&=A_{k,p}P_{k-1,p}A_{k,p}^T+A_{k,pv}P_{k,v}A_{k,pv}^T+Q_{k,p}\\&\approx P_{k-1,p}+A_{k,pv}P_{k,v}A_{k,pv}^T+Q_{k,p}\end{aligned}$$

Corrección Global:

$$K_{k,p} = P_pH_p^T\left(H_pP_pH_p^T + R_p\right)^{-1}$$

$$\hat{x}_{k,p} = \hat{x}_{k,p}+K_{k,p}\left(z_{k,p}-H_{k,p}\hat{x}_{k,p}\right)$$

$$P_{k,p} = P_{k,p} - K_{k,p}H_{k,p}P_{k,p}$$

Algoritmo 7: Algoritmo KF recursivo en cascada con modelos local/global en cascada, asignación de entradas, predicción local de velocidades, modelo global de postura y corrección global continua

Entrada: u_{k-1}=u_{Akns}=$[\, u_1 \;\; u_3 \,]^T$=u_{dif}=$[\, a \;\; \alpha \,]^T$,$\hat{x}_{k-1,v}$,$P_{k-1,v}$,$\hat{x}_{k-1,p}$,$P_{k-1,p}$

Medición: $z_{k,v} = [\; v_{x,enc} \;\; v_{y,enc} \;\; \omega_{enc} \;\; \omega_{gyr} \;\; \omega_{comp} \;\;]^T_k$,
$z_{k,p} = [\; x_k \;\; y_k \;\; \theta_k \;\;]^T_{GP}$

Datos: A_k *y* B_k *del modelo lineal local* (B.23) *(o* (B.52)*)*,$Q_{k,v}$,$R_{k,v}$,
$f_p(\cdot)$ *y* $h_p(\cdot)$ *del modelo no lineal global* (B.11),$Q_{k,p}$,$R_{k,p}$

Salida: $\hat{x}_{k,v} = \left[\; v_x \quad v_y \quad \omega \;\right]^T_k$,$\hat{x}_{k,p} = \left[\; x \quad y \quad \theta \;\right]^T_k$,$P_{k,p}$, $P_{k,v}$

Inicialización: $\hat{x}_{0,v}$,$P_{0,v}$,$\hat{x}_{0,p}$,$P_{0,p}$

Para *el instante actual* k **hacer**

Predicción Local:

$$\hat{x}_{k,v} = A_{k-1,v}\hat{x}_{k-1,v} + B_{k-1,v}u_{k-1}$$
$$P_{k,v} = A_{k-1,v}P_{k-1,v}A^T_{k-1,v} + W_{k-1,v}Q_{k-1,v}W^T_{k-1,v}$$

Corrección Local:

$$K_{k,v} = P_{k,v}H^T_{k,v}\Big(H_{k,v}P_{k,v}H^T_{k,v} + N_{k,v}R_{k,v}N^T_{k,v}\Big)^{-1}$$
$$\hat{x}_{k,v} = \hat{x}_{k,v} + K_{k,v}\left(z_{k,v} - H_{k,v}\hat{x}_{k,v}\right)$$
$$P_{k,v} = \left(I - K_{k,v}H_{k,v}\right)P_{k,v}$$

Modelo Global, Covarianza Global:

$$\hat{x}_{k,p} = f_p\left(\hat{x}_{k-1,p}, \hat{x}_{k,v}\right)$$
$$A_{k,p} = \frac{\partial f_p}{\partial x_p}\Big|_{\substack{x_{k-1}\\u_{k-1}}}, A_{k,pv} = \frac{\partial f_p}{\partial x_v}\Big|_{\substack{x_{k-1}\\u_{k-1}}}$$
$$P_{k,p} = A_{k,p}P_{k-1,p}A^T_{k,p} + A_{k,pv}P_{k,v}A^T_{k,pv} + Q_{k,p}$$
$$\approx P_{k-1,p} + A_{k,pv}P_{k,v}A^T_{k,pv} + Q_{k,p}$$

Corrección Global: $H_{k,p} = \frac{\partial h_p}{\partial x_p}\Big|_{\hat{x}_k,0}$ =Identidad 3×3 según ecuación (3.20)

$$K_{k,p} = P_{k,p}H^T_{k,p}\Big(H_{k,p}P_{k,p}H^T_{k,p} + R_{k,p}\Big)^{-1}$$
$$\hat{x}_{k,p} = \hat{x}_{k,p} + K_{k,p}\left(z_{k,p} - H_{k,p}\hat{x}_{k,p}\right)$$
$$P_{k,p} = P_{k,p} - K_{k,p}H_{k,p}P_{k,p}$$

fin

Una vez expuestas las distintas versiones de los algoritmos de fusión desarrollados hasta el momento, se realiza una serie de pruebas en simulación para medir los tiempos de ejecución requeridos por cada uno.

Estas pruebas se realizan mediante el programa Matlab® ejecutado en un computador de 2.4Ghz y 4GB RAM. Se lanzan los distintos algoritmos de localización para calcular la posición del robot mientras realiza o recorre una trayectoria cuadrada. Aproximadamente cada prueba tuvo una duración de 30 segundos utilizando un periodo de muestreo de $50ms$, es decir, aproximadamente cada ejecución realizó 600 iteraciones y la tabla 3.1 muestra un resumen de las pruebas y el tiempo promedio de ejecución de cada algoritmo.

Tabla 3.1: Pruebas realizadas en simulación

Tipo de filtro	Algoritmo	Estado x_k	Medición z_k	Salida	Tiempo(ms)
EKF	Alg. 1	$[x_{k,p}\ \ x_{k,v}\ \ u_k]^T$	$[z_{k,p}\ \ z_{k,v}\ \ u_k]^T$	$x_{k,p}$	0.138
UKF	Alg. 2	$[x_{k,p}\ \ x_{k,v}\ \ u_k]^T$	$[z_{k,v}\ \ u_k]^T$	$x_{k,p}$	1.160
EKF v.2	Alg. 3	$[x_{k,p}\ \ x_{k,v}]^T$	$[z_{k,p}\ \ z_{k,v}]^T$	$x_{k,p}$	0.089
UKF v.2	Alg. 4	$[x_{k,p}\ \ x_{k,v}]^T$	$z_{k,v}$	$x_{k,p}$	0.665
EKF v.3	Alg. 5	$[x_{k,p}\ \ x_{k,v}]^T$	$z_{k,v}$	$x_{k,p}$	0.061
EKF v.4	Alg. 6	$[x_{k,p}\ \ x_{k,v}]^T$	$z_{k,v}$	$x_{k,p}$	0.059
KF	Alg. 7	$x_{k,v}$	$z_{k,v}$	$x_{k,p}$	0.029

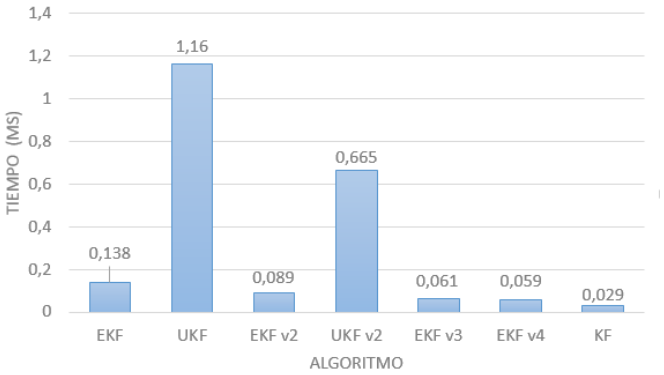


Figura 3.4: Tiempo promedio de ejecución, simulación mediante Matlab en un ordenador, gráfica comparativa entre los distintos algoritmos implementados

Según los resultados obtenidos, la reducción mediante la reasignación de entradas y el vector de medición minimiza aproximadamente hasta un 60 % el coste computacional total del UKF y EKF, luego la metodología ofrece buenos resultados. También se observa claramente en el gráfico 3.4 una diferencia notable entre tiempos de ejecución. El KF es el filtro más rápido, seguido por el EKF v4 y EKF v3. Por

otro lado, el filtro UKF v2 continúa requiriendo un alto coste temporal de ejecución comparado con el resto de algoritmos.

3.3 Resultados Obtenidos en Pruebas Empíricas

El siguiente paso consiste en obtener los resultados de la implementación del filtro en plataformas con recursos limitados. A continuación se describe la implementación de los algoritmos de fusión sensorial de corrección continua en los robots LEGO Mindstorms NXT, Summit XL y Rbcar.

3.3.1 Plataforma LEGO Mindstorms NXT

En enero de 2006 se presentó en el International Consumer Electronics Show la segunda generación de robots Lego: el Lego Mindstorms NXT. La nueva versión además de otros cambios menores en los sensores electrónicos y las piezas de construcción, incorpora una unidad de control nueva: el NXT. Esta nueva unidad de control, está basada en un microcontrolador de 32 bits: ARM7, con 256 Kbytes FLASH y 64 Kbytes de memoria RAM.

Para la programación y las comunicaciones, el NXT está equipado con un puerto USB 2.0 y con un dispositivo inalámbrico Bluetooth clase II, V2.0. Además, también incorpora una pantalla LCD gráfica matricial de 100x64 píxeles, un altavoz de sonido real y 4 botones que permiten una programación simple gracias a un entorno muy intuitivo basado en iconos.

Así mismo, el NXT dispone de 4 entradas (una de ellas incluye una expansión IEC 61158 Type 4/EN 50 170 para usos futuros) y 3 salidas analógicas, por lo que, además de disponer de 1 entrada más que en la versión anterior y dado que los sensores de rotación están completamente integrados en los actuadores eléctricos en esta versión, es posible tener conectados un número mayor de dispositivos sensores.

Desde su aparición, han surgido diversos firmwares para el control del módulo NXT. En esta tesis se ha trabajado principalmente con el firmware LeJOS el cual permite la programación del NXT en lenguaje JAVA. Además, LeJOS permite la implementación de múltiples hilos de ejecución (programación multithreading) en la unidad de control, diversas librerías que implementan funciones matemáticas básicas, manejo en alto nivel de sensores y actuadores, administración de las comunicaciones Bluetooth, y otras muchas funcionalidades.

El robot LEGO NXT es una plataforma de recursos limitados, principalmente en cuanto a memoria disponible siendo capaz de incorporar únicamente 255 variables locales, 1024 constantes y una longitud máxima del código de programación de $64kb$.

Al igual que pasaba con la versión anterior, combinando los bloques de construcción, la fácil programación del NXT y su interfaz de entrada y salida se puede obtener un sistema de prototipado rápido para el desarrollo de una gran variedad de actividades, lo que ha permitido que este sistema haya sido ampliamente aceptado como una herramienta para la investigación y la educación universitaria.

Para esta tesis destaca la construcción de dos modelos distintos de robots, uno de configuración diferencial y otro de configuración Ackerman, expuestos a continuación.

- El robot diferencial construido utilizando el LEGO NXT se muestra en la figura 3.5. Los sensores embarcados que se indican, son los utilizados en el esquema de fusión definido en el presente capítulo y corresponden a dos acelerómetros ubicados en la plataforma justo sobre las ruedas del robot y alineados con los ejes de las mismas, además de un giróscopo, una brújula y los encoders de los motores del robot.

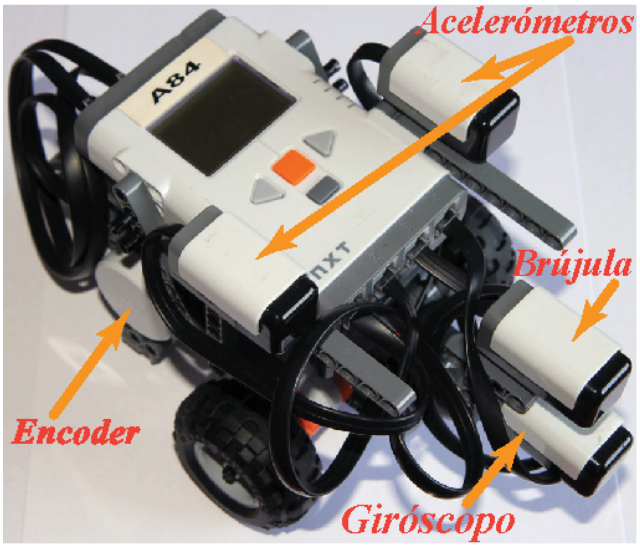


Figura 3.5: Robot móvil diferencial contruido con el LEGO NXT

- De forma similar, el robot Ackerman construido con el LEGO NXT se muestra en la figura 3.6. Nuevamente, los sensores a utilizar en el esquema de fusión se indican en esta figura y corresponden a dos acelerómetros ubicados en la plataforma justo sobre el centro de los ejes de las ruedas del robot, además de un giróscopo, una brújula y los encoders de los motores trasero y delantero del robot.

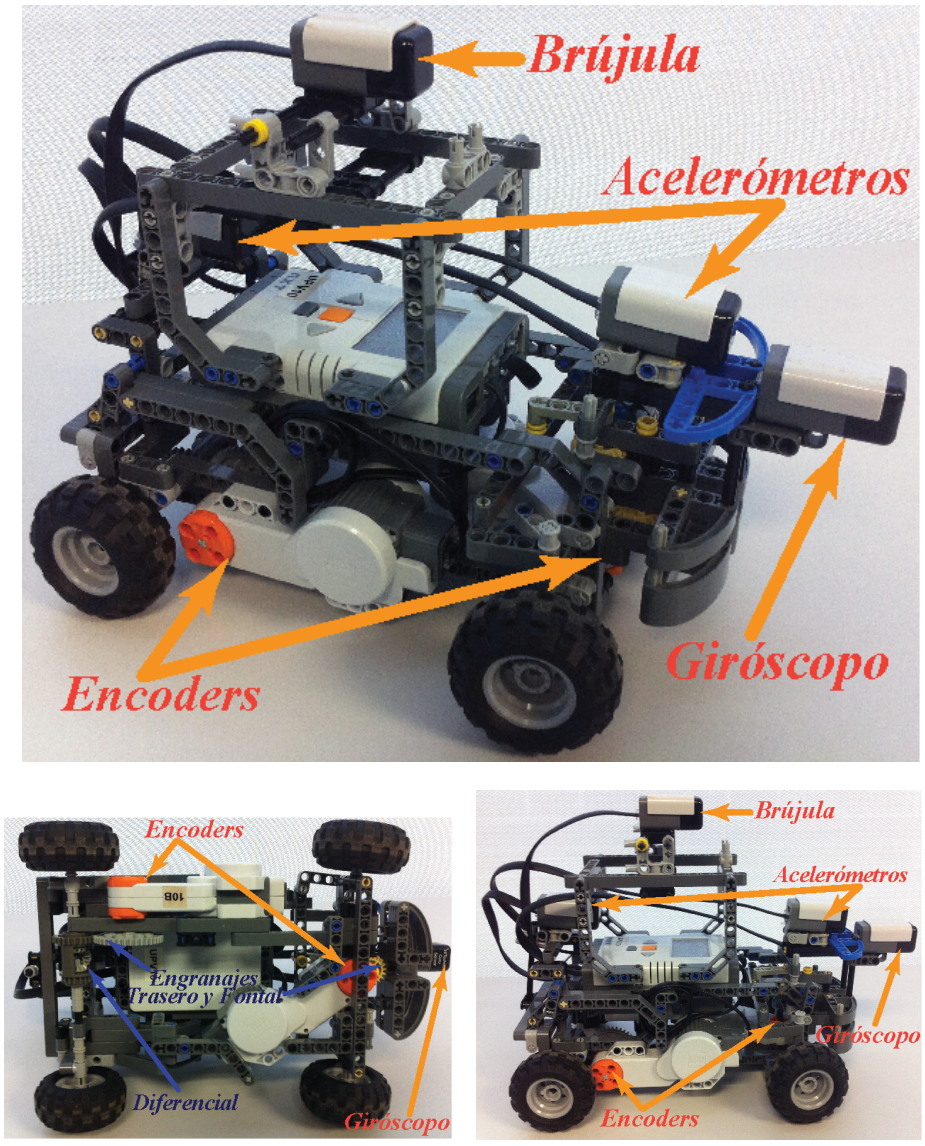


Figura 3.6: Robot móvil en configuración Ackerman construido con LEGO NXT

Cabe destacar que se han agregado dos engranajes a la salida del motor delantero, para incrementar el rango de movimiento y permitir un mejor control de la dirección del vehículo.

Además, se emplea un diferencial en la transmisión del motor trasero el cual permite transmitir el movimiento del motor a las ruedas traseras a la vez que se permite la rotación de ambas a distintas velocidades (siendo v_x el promedio de ambas), lo que permite el giro adecuado del robot al negociar una curva.

3.3.2 Resultados en la Plataforma Experimental LEGO Mindstorms NXT

Desde un primer momento la viabilidad de poder implementar cualquiera de las dos versiones del filtro UKF ha sido mermada por el alto número de operaciones requeridas y el alto número de variables necesarias. Entre las muchas limitaciones que restringen la programación del robot LEGO NXT con el firmware leJOS, se encuentra un espacio de trabajo donde un método no puede superar los 65535 bytes, el compilador no dispone de operaciones aritméticas complejas y los vectores están limitados a 511 posiciones. No es posible cumplir estas restricciones para la implementación del UKF. Sin embargo sí que lo es con las versiones del EKF v3, EKF v4 y el filtro KF.

Se realizó una ardua tarea de reducción de código y reutilización de variables para conseguir implementar los filtros bajo un bucle de control deseado de 50ms ($T_s = 50$ms). Se optimizaron las operaciones de inversión de matrices y se redujo al máximo los tiempos de comunicaciones entre el procesador y el hardware de sensorización externo. Para el caso de los acelerómetros por ejemplo, hubo que escribir un nuevo controlador bajo el protocolo I2C a bajo nivel porque el que venía con la API del software oficial de LeJOS no estaba optimizado y ralentizaba su lectura.

Se realizó entonces una prueba de 1 minuto de duración, en la cual el robot realizaba una trayectoria cuadrada únicamente con la información ofrecida por los sensores embarcados (sin la ayuda de un posicionamiento global externo). Se midieron los tiempos dedicados a las distintas fases del filtrado: Tiempo de lectura de sensores incluyendo calibrado y preprocesamiento ($TRead$), el tiempo dedicado exclusivamente a la ejecución del filtro ($TKalman$), el tiempo dedicado a la ejecución del algoritmo de control de navegación y seguimiento de trayectoria ($TControl$) y el tiempo dedicado a escribir en un fichero de texto las variables requeridas para la supervisión de las demostraciones ($TWrite$).

Dada la variabilidad de los tiempos de ejecución entre distintas pruebas, la figura 3.7 muestra tres casos por cada prueba: el caso a representa el tiempo promedio de la ejecución de la tarea por ciclo (cada T_s), el caso b se corresponde con el tiempo máximo de ejecución por ciclo medido durante toda la prueba y el caso c muestra el peor caso para cada fase del filtrado que se ha dado durante toda la prueba.

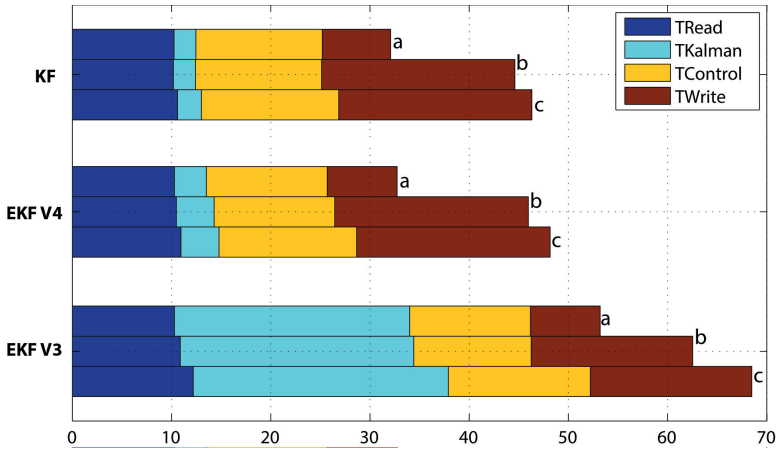


Figura 3.7: Tiempos de ejecución de los algoritmos de fusión sensorial en la unidad de control NXT durante 1min, robot diferencial, caso *a*: tiempo promedio, caso *b*: tiempo de la peor ejecución y caso *c*: tiempos máximos de cada tarea

Cabe destacar que los peores casos de cada fase no tienen por qué ocurrir en el mismo ciclo de ejecución.

Los resultados de la figura 3.7 muestran que únicamente los algoritmos *KF* y *EKF V4* cumplen el requisito de no sobrepasar en ningún caso el periodo de muestreo deseado de 50 ms. Por otro lado, aún en el mejor de los casos, el algoritmo *EKF V3* no consigue obtener un tiempo de ejecución adecuado al tiempo de ciclo por lo que se descarta su implementación.

Entre los algoritmos *KF* y *EKF V4* la diferencia del coste de ejecución puede parecer no ser significativa, sin embargo, el KF tarda entre 4 y 5 ms menos que el *EKF V4* y su implementación resulta más sencilla en términos de programación por lo que ha sido el algoritmo finalmente escogido para realizar las pruebas empíricas que se describen a continuación.

Para validar el filtro KF implementado se realizaron varias pruebas donde el robot LEGO NXT en configuración diferencial debía seguir 4 trayectorias distintas: un cuadrado, un círculo, una trayectoria en forma de Lemniscata y otra en forma de Rosa Polar tal y como muestran las figuras 3.8,3.9 y 3.10.

Con el objetivo de poder medir la precisión y el desempeño de la localización del algoritmo, se instaló un señuelo encima del robot y mediante una cámara cenital en lo alto del escenario se capturaba y se posicionaba el robot durante toda la prueba. Esta monitorización de la prueba se realizó mediante un procesamiento externo en un ordenador dedicado y tan sólo se utilizó para poder comparar la localización calculada internamente por el robot, con la calculada mediante

la cámara. Este procedimiento permite visualmente verificar el desempeño del algoritmo y ponderar su precisión cuantitativamente. En azul se muestra los datos de odometría calculados por el robot internamente y en rojo la trayectoria real capturada por la cámara.

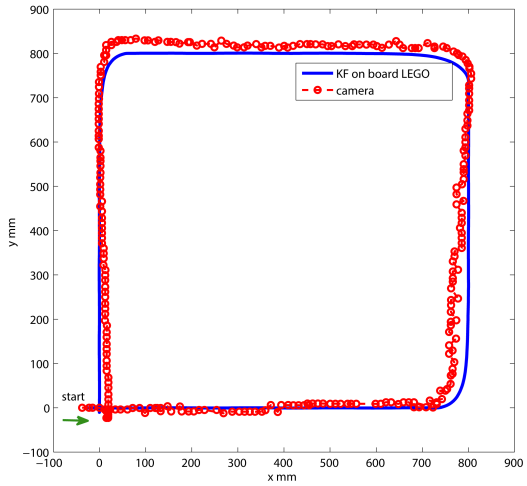


Figura 3.8: Desempeño de los algoritmos de localización y navegación, KF implementado en el robot diferencial, trayectoria de referencia cuadrada

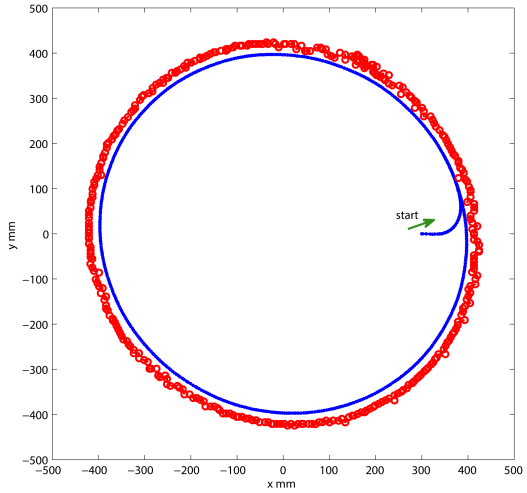
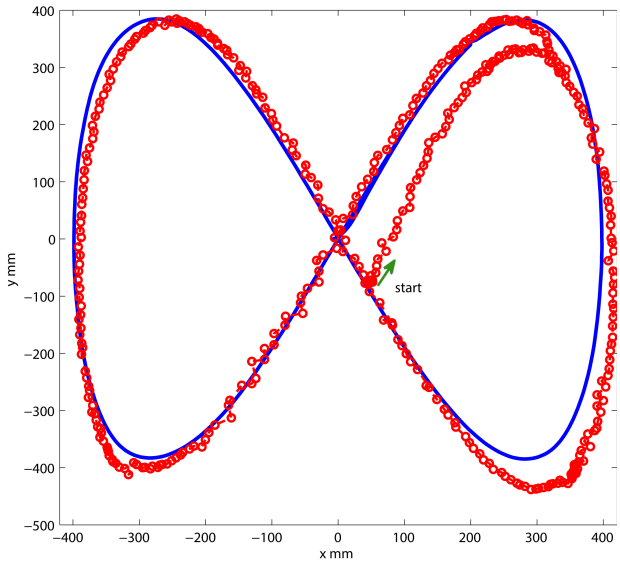
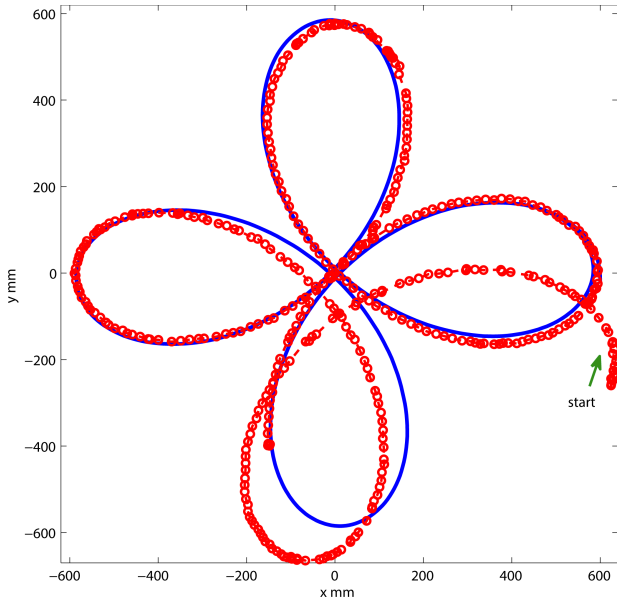


Figura 3.9: Desempeño de los algoritmos de localización y navegación, KF implementado en el robot diferencial, trayectoria de referencia circular



(a) Lemniscata



(b) Rosa Polar

Figura 3.10: Desempeño de los algoritmos de localización y navegación, KF implementado en el robot diferencial, trayectorias de referencia lemniscata y rosa polar

Para poder comparar visual y empíricamente los resultados del uso del filtro de Kalman respecto a un control básico PID basado únicamente en la lectura del valor de los encoders de las ruedas (odometría) y calibrados mediante [121], se realizó una prueba de larga duración (30 min) para una trayectoria cuadrada cuyos resultados se muestran a continuación.

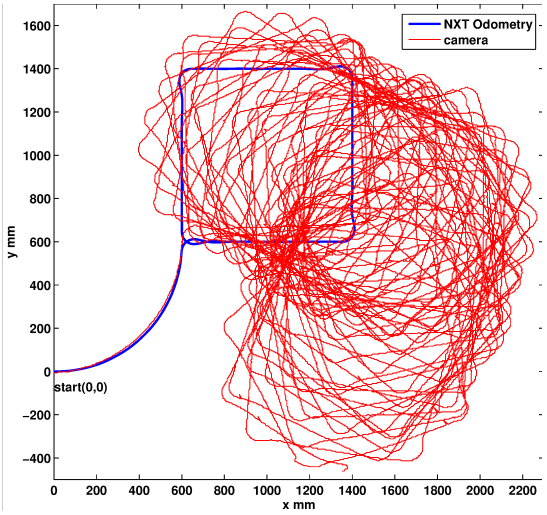


Figura 3.11: Prueba de larga duración (28:54 min) con odometría basada únicamente en la lectura de los encoders

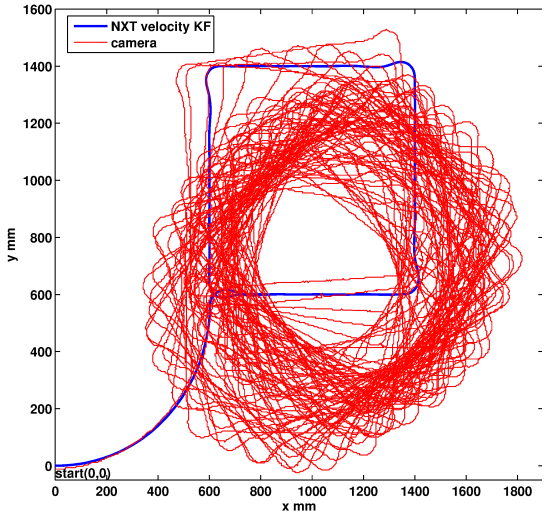


Figura 3.12: Prueba de larga duración (>30 min) con localización basada en el KF

La trayectoria de color azul representa la estimación calculada por el robot durante la primera vuelta (1 minuto aproximadamente), ya que debido a las limitaciones de la memoria no es posible almacenar más datos dentro del robot. La trayectoria en color rojo se corresponde con la monitorización de la posición del robot por parte de la cámara cenital.

El límite de duración de la prueba se determinó por el tiempo que el robot era capaz de mantenerse dentro del escenario gobernado por la cámara cenital. En la figura 3.11 se muestra como el control mediante odometría va desviándose poco a poco del centro del escenario hasta que finalmente sale del mismo en el minuto 28:54. A medida que el error de estimación se acumula, la postura del mismo va divergiendo. El error de estimación no es tenido en cuenta por el robot, quien considera que su postura es adecuada y que está siguiendo correctamente la trayectoria aunque en realidad su postura es errónea según se puede observar con la monitorización de la cámara. En cambio, la figura 3.12 muestra cómo el control mediante el KF consigue mantener la trayectoria durante más de 30 minutos aunque no es capaz de mantener la orientación de la misma. Esto es debido a que el sensor de la brújula se tuvo que ponderar con muy poco peso dentro de la entrada del algoritmo KF ya que en el escenario del laboratorio existían unas perturbaciones magnéticas importantes que perjudicaban seriamente las mediciones del sensor. Además, al carecer de un posicionamiento global, la postura también diverge pero más lentamente.

La figura 3.13 muestra una comparativa del error promedio de ambas pruebas.

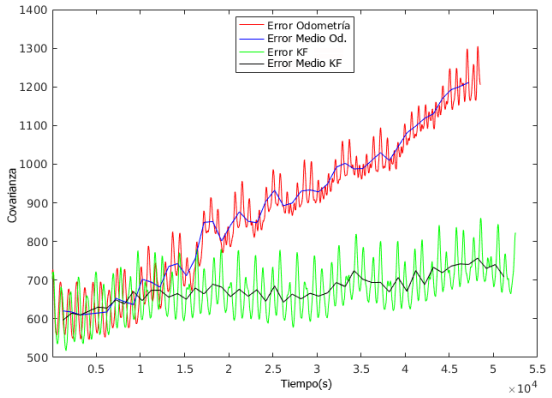


Figura 3.13: Comparativa del error de localización promedio de la prueba de larga duración (>30 min) con localización basada en odometría y KF

El error del experimento por odometría presenta un error casi exponencial mientras que el experimento con la incorporación del filtro de Kalman presenta un error lineal con una mínima pendiente.

Del mismo modo, se realizaron pruebas empíricas con el robot NXT de configuración Ackerman. La figura 3.14 muestra un ejemplo del desempeño del control de odometría (sólo encoders) para el seguimiento de una trayectoria cuadrada.

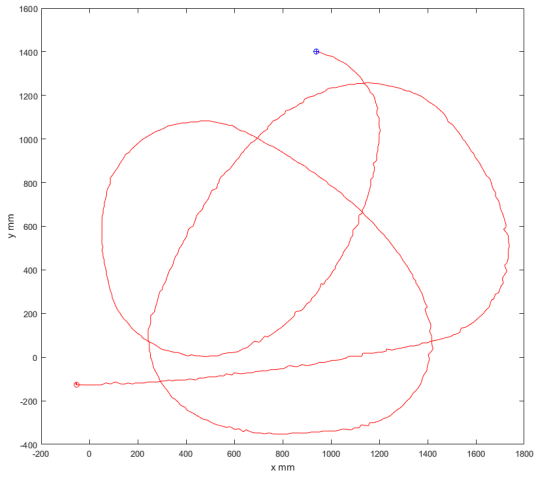


Figura 3.14: Robot LEGO NXT de configuración Ackerman realizando una trayectoria cuadrada únicamente con la medida de los encoders (odometría)

Naturalmente la configuración Ackerman restringe el ángulo de giro en las esquinas por lo que aparece el efecto de recorte mucho más acentuado que con el robot en configuración diferencial y la trayectoria resultante se asemeja más a un círculo que a un cuadrado. Además, el robot sale del escenario en el segundo 0:24 habiendo conseguido tan sólo un par de trayectorias completas.

En la figura 3.15 se muestra la ejecución de la misma trayectoria utilizando el algoritmo KF para el posicionamiento del NXT. En este caso se observa una mejora del seguimiento de la trayectoria aunque al igual que con el control por odometría, la trayectoria del cuadrado se ve deformada en las esquinas. El algoritmo, sin embargo, es capaz de mantener al robot en el centro del escenario durante los 30 minutos que duró la prueba. Por otro lado, es evidente que se produce el mismo efecto del error de orientación acumulativo que ocurre con la configuración diferencial.

Como conclusión cabe destacar la mejora notable del seguimiento de trayectorias de los robots al utilizar el algoritmo KF desarrollado. Tanto en configuración diferencial como en configuración Ackerman, se ha conseguido con esta aportación mejorar el desempeño y la fidelidad de la localización local de los robots LEGO Mindstorms NXT.

Llegado a este punto se plantea la aportación de la implementación del filtro de Kalman aplicado a sistemas distribuidos.

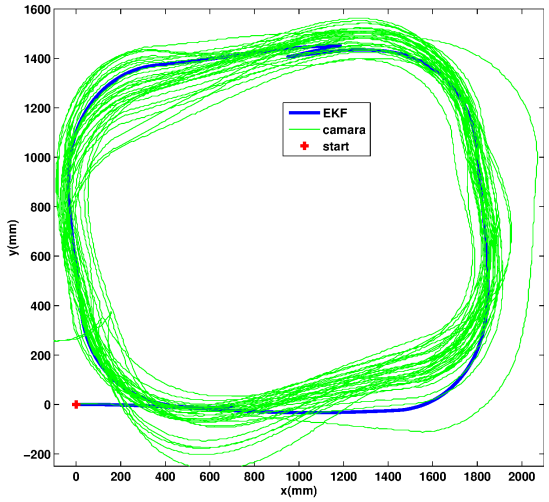


Figura 3.15: Robot LEGO NXT de configuración Ackerman realizando una trayectoria cuadrada mediante el algoritmo KF durante 30 min.

3.3.3 Plataformas Summit XL y Rbcar

A continuación se describen las plataformas basadas en sistemas distribuidos que se han utilizado para aplicar e integrar los algoritmos desarrollados y recopilar y analizar los resultados.

3.3.3.1 Summit XL

El robot Summit XL (Fig. 3.16) ha sido desarrollado por la empresa Robotnik. Se trata de una plataforma móvil en configuración diferencial basada en cuatro motores brushless independientes. Tiene un peso aproximado de 30Kg, soporta una carga máxima de 20Kg adicionales, alcanza una velocidad máxima de 3m/s y tiene una autonomía de 180 minutos.

Su cerebro se compone de una tarjeta mini ITX con un procesador Intel Atom CPU N2800 a 1.86GHz, una memoria DDR3 de 2GB y un disco duro de estado sólido de 128GB. Ofrece las conexiones típicas de una placa mini ITX como son puertos USB, Ethernet, salida VGA y HDMI e incorpora también un router configurado como punto de acceso para conectarse al robot vía WiFi. El sistema operativo que viene de fábrica es un Linux Ubuntu LTS. Además el robot viene configurado de fábrica para ser integrado completamente en ROS. Ofreciendo tanto los modelos urdf y xacro del mismo para sus simulaciones en Gazebo o rviz, como los controladores básicos de odometría y teleoperación.

Figura 3.16: Robot móvil Summit XL con Hokuyo URG-04LX-UG01

Para la detección del entorno se ha integrado en el robot, un láser 2D modelo Hokuyo URG-04LX-UG01. Este dispositivo se sitúa entre los range-finder láser más pequeños del mercado gracias a que pesa menos de 160 gramos y consume 2.5 vatios. Está específicamente diseñado para aplicaciones de interior y permite medir hasta 5.6 metros con un ángulo de visión de 240º, ofreciendo unas medidas fiables con una resolución aproximada de 0.352º. ROS ofrece un paquete para gestionar este modelo de láser con cualquier plataforma, lo cual facilita mucho su integración. Además, en las aplicaciones de exteriores se ha integrado un sensor GPS tal y como se describe en la sección 5.3.2.

3.3.3.2 Rbcar

La plataforma móvil Rbcar (imagen 3.17) consiste en un coche eléctrico con configuración cinemática Ackerman de tracción. La tracción es controlada mediante un motor AC con encoder incremental y la dirección mediante un sistema de servodirección con encoder absoluto.

Figura 3.17: Robot Rbcar construido por la empresa Robotnik

Ha sido desarrollado por la empresa Robotnik. Gracias a su estructura mecánica y el cajón trasero, puede transportar altas cargas así como cualquier tipo de sensor. Con la configuración adecuada el robot puede navegar autónomamente, ser tele-operado con un joystick o con un volante, a modo de vehículo eléctrico.

Lleva equipados frenos de tambor delanteros y traseros que permiten detener el robot inmediatamente, opcionalmente mediante una seta remota de emergencia. Dispone de un ordenador a bordo con un PC empotrado con Linux (Intel Dual Core y 4 Gb de RAM) y un router con el que crea una red a la que el usuario se puede conectar para poder controlar y realizar cambios en la configuración del coche. El ordenador está conectado a una pantalla táctil con la que el usuario puede también interactuar a la hora de ejecutar los diversos programas.

El Rbcar pesa 350 Kg sin baterías y posee una capacidad de carga máxima de $250Kg$. Alcanza una velocidad máxima de $30km/h$.

El software de la plataforma funciona bajo Linux y bajo la arquitectura ROS. Se ha equipado con distintos lasers y dispositivos de posicionamiento GPS para realizar las distintas pruebas de esta tesis. Es sin duda la plataforma con mayores recursos que se ha utilizado para las pruebas experimentales.

3.3.4 Resultados en Sistemas Distribuidos

Como ya se ha comentado previamente, los sistemas distribuidos utilizados en el presente trabajo utilizan el protocolo de publicación-suscripción para gestionar las comunicaciones de datos entre procesos y la especificación FIPA-ACL para las comunicaciones de alto nivel de abstracción entre nodos. En el caso de la localización local de un robot, el proceso de integración de algoritmos de fusión sensorial va a utilizar expresamente la metodología de mensajes de publicación-suscripción o topics ya que en la mayoría de los casos todos los procesos se ejecutan localmente dentro del mismo robot.

A diferencia de los sistemas centralizados donde el bucle de control ejecuta secuencialmente, (la lectura de los sensores, el procesamiento de la información y la ejecución del control), en los sistemas distribuidos las tareas están desacopladas de modo que cada una se ejecuta de manera independiente al resto. Por ejemplo, para cada sensor, se existe un proceso o nodo, que gestiona el acceso al hardware del sensor y obtiene y procesa la información recogida por el mismo, pudiendo publicarla a cierta frecuencia por el canal de comunicación del sistema a modo de tópics. Esta es la filosofía seguida para la integración de algoritmos de fusión sensorial en sistemas distribuidos.

En esta parte del trabajo se han utilizado principalmente dos plataformas: el robot *Summit XL* en configuración diferencial (presentado en la sección 3.3.3.1) y la plataforma *Rbcar* en configuración Ackerman (presentada en la sección 3.3.3.2),

desarrolladas las dos por la empresa valenciana Robotnik. Las plataformas incorporan el RSF de ROS, al cual se le ha añadido el middleware JADE-ROS desarrollado en la presente tesis. Los robots tienen instalado un encoder en cada rueda (y en la dirección del volante en el caso del *Rbcar*), un giróscopo y en este trabajo se integró una unidad de masa inercial modelo RLVBIMU03 de la marca Racelogic con el objetivo de mejorar la localización local de los vehículos.

Puesto que la arquitectura es la misma en los dos vehículos, los dos comparten la ejecución de los mismos paquetes y prácticamente el mismo software desarrollado para la integración sensorial. Lo único que cambia entre ambos se reduce a la descripción de la cinemática, la cual se especifica en el paquete *robot_description* y el desarrollo del control básico de posición de cada uno del paquete *controller*.

Dentro de cada sistema, cada sensor se gestiona de manera individual como un nodo independiente, resultando un total de cuatro nodos de tipo driver para el manejo de los encoders de las ruedas (uno más para el volante en el caso del *Rbcar*) un nodo para la lectura del giróscopo y otro nodo para la gestión de la IMU. Cada uno de estos nodos resuelve el acceso al hardware pertinente y ofrece los datos recogidos mediante la publicación por topics.

Para el cálculo de una odometría base sobre la que poder comparar los resultados, el nodo *controller* se suscribe a los tópics de los encoders de las ruedas y al tópic del giróscopo (*/gyro*) tal y como muestra el siguiente esquema:

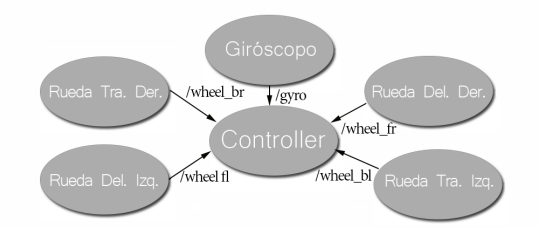


Figura 3.18: Esquema de conexión al nodo *controller*

El nodo *controller* de las dos plataformas ofrece un servicio de inicialización de la odometría llamado *set_odometry*. Mediante este servicio se puede inicializar la posición local del robot y se calcula el desplazamiento relativo en cada iteración de control actualizando los valores de odometría. La frecuencia de dicho bucle se ha establecido en 50 Hz, es decir, que cada 20 ms el nodo utiliza los últimos valores recibidos por los topics de los encoders y el giróscopo para estimar el desplazamiento que ha sufrido el robot respecto a la última iteración. Como resultado, el mismo nodo publica un tópic de tipo *Odometry* con los nuevos valores de posición, velocidad angular y lineal del robot. Además el tipo de dato publicado incorpora un campo donde se especifica la covarianza estimada para cada uno de los valores calculados.

Esta odometría base será la que se compare con la odometría resultante de integrar el sensor de la IMU en el proceso de localización local mediante fusión sensorial. Para este proceso se utiliza la implementación de un filtro extendido de Kalman en un único nodo. Este nodo se suscribe a los topics de los sensores y en cada iteración ofrece por otro tópic su propio cálculo de la odometría corregida.

El RSF de ROS, está orientado hacia la reutilización de código compartido por desarrolladores de todo el mundo y es por eso que los paquetes deben ser flexibles y aplicables a cualquier tipo de robot. Es por este motivo por el que la entrada del vector de estados del algoritmo se define para cualquier configuración como:

$$\boldsymbol{x}_k = \begin{bmatrix} x & y & z & \alpha & \beta & \gamma & x' & y' & z' & \alpha' & \beta' & \gamma' & x'' & y'' & z'' \end{bmatrix}_k^T \quad (3.29)$$

El script de lanzamiento de los nodos contiene unos parámetros de configuración sobre los cuales se define la especificación de cada robot. Para el caso de la integración del Summit XL, el nodo del EKF se suscribe a la odometría base publicada por el nodo *controller* y al tópic */imu* publicado por la IMU. En ambos casos es necesario especificar las características de las fuentes de datos mediante los parámetros de configuración. Para la odometría del Summit XL y el Rbcar se especifica únicamente la posición X, Y y la posición del yaw, es decir, la orientación. Para los datos de la IMU, la configuración que mejores resultados ofrece es la velocidad angular de giro en el eje Z.

De este modo se integra la información de los dos tópics y la nueva localización ofrecida por el algoritmo EKF resultado de la fusión, se publica a través de un tópic nuevo de odometría */odom_filtered*. La relación entre topics se muestra en la figura 3.19.

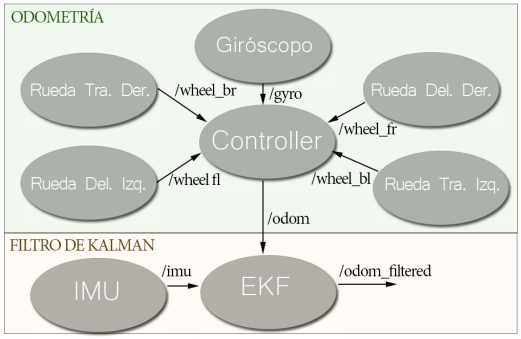


Figura 3.19: Esquema de conexión entre los nodos de la odometría y el KF.

Para las pruebas empíricas se realizaron distintos tests de ejecución. Se utilizó el entorno *rviz* para obtener la visualización de los datos comparando la odometría

calculada por el nodo *controller* con la odometría calculada por el nodo *ekf*. Los resultados para una trayectoria cuadrada se muestran en la figura 3.20.

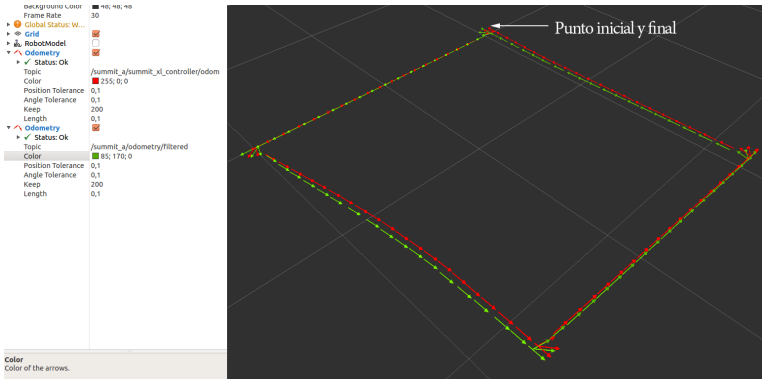


Figura 3.20: Trayectoria cuadrada con ekf(verde) y *odometría*(rojo)

La trayectoria de color rojo muestra la salida calculada por la odometría, mientras que la verde muestra la obtenida mediante el KF. Aunque la diferencia puede no parecer significativa en la imagen 3.20, esto es debido a que se trata de una prueba de corta duración realizada a velocidad reducida. Cuantificar los resultados en este caso resulta complejo dado que no hay una referencia o una fuente de información externa que pueda localizar el vehículo de manera global, de modo que una de las pruebas realizadas para verificar que la localización con el nodo ekf contra la del nodo *controller* era mejor, fue realizar una trayectoria aproximada en forma de ocho cuyos puntos inicial y final fuesen el mismo punto visualmente.

Esta prueba se muestra en la figura 3.21, donde se puede observar que en el inicio la salida de los dos nodos es la misma pero conforme se produce el movimiento ambas trayectorias divergen poco a poco. Concretamente, se puede comprobar que al terminar la trayectoria, el último punto de la línea verde (ekf) se sitúa más cerca del comienzo que el último de la línea roja (*odometría*). De ser perfectos, ambos algoritmos deberían haber posicionado el robot justo en el mismo punto en el que empezó. Sin embargo, en ese último punto el ekf obtiene un error de estimación de posición de 5 mm mientras que la *odometría* supera los 12 mm.

Como se ha comentado anteriormente, dadas las dimensiones de los dos vehículos (summit y rbcar) resulta complicada la monitorización de su posición mediante un agente externo como en el caso del Lego Mindstorms NXT, por este motivo se realizó una trayectoria donde el principio y el final era el mismo. A la vista de los resultados se observa una mejora importante de la localización local tanto del Summit XL como del Rbcar, sobre todo en cuanto al cálculo de la orientación. Los encoders de las ruedas responden bien a los cambios de velocidad y el posicionamiento X Y es bastante preciso siempre que no exista deslizamiento.

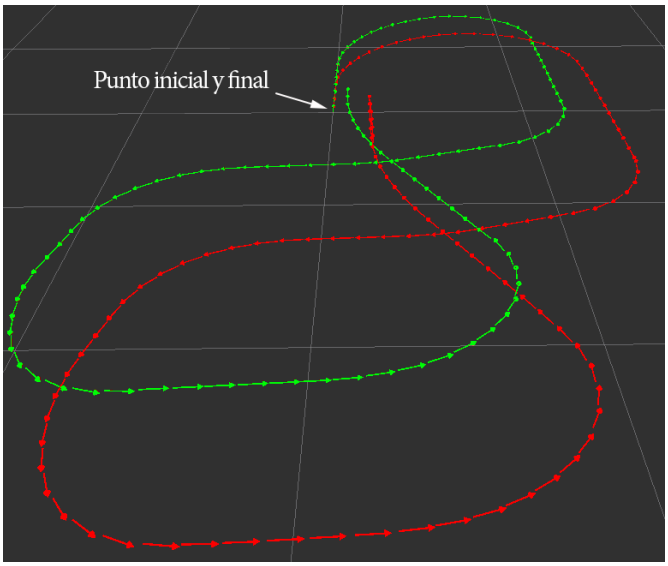


Figura 3.21: Trayectoria en forma de ocho con *ekf*(verde) y *odometría*(rojo)

Se han probado distintas configuraciones para la configuración del nodo *ekf*. Sin embargo la mostrada en esta sección es la que mejores resultados ha ofrecido. El mayor problema de esta integración, resultó ser una error de correspondencia entre las frecuencias de publicación de los topics. El problema del sincronismo de los datos una problemática típica en sistemas desacoplados. En este caso hubo que modificar el código original que incorporaba el robot Summit XL para que el giróscopo publicase sus medidas con mayor frecuencia. Para mayor detalle, existe un hilo en lo referente a esta problemática en el sitio web de ROS answers, `/http://answers.ros.org/question/198451/integrate-an-imu-sensor-with-robot_localization/`.

Del mismo modo que ocurría con la implementación en la plataforma de recursos limitados LEGO Mindstorms NXT, la localización local del Summit XL y el Rbcar mediante la fusión sensorial también empeora cuanto mayor es el desplazamiento. En ambos casos llega un momento en el que el error de posicionamiento es demasiado alto y muy poco fiable como para mantener el robot de manera autónoma en un escenario desconocido. Esto es un error muy común en la problemática de la localización local de robots. Es debido a que el poco error que se pueda cometer en cada iteración de la localización, se acumula y realimenta directamente el bucle de control por lo que a la siguiente iteración el error inicial es superior al anterior. Para evitar esto es necesario dar el salto hacia una localización global, donde no sólo la posición inicial del robot es el origen de coordenadas si no que existe otro origen de coordenadas externo sobre el que el robot se posiciona. Este es el objetivo y la problemática que se aborda en el siguiente capítulo.

3.4 Conclusiones del capítulo

En este capítulo se han expuesto las aportaciones y contribuciones de la presente tesis a la mejora de la autolocalización local de robots móviles con recursos limitados mediante el uso de filtros de fusión sensorial:

- Se han expuesto las distintas estrategias de fusión sensorial aplicadas al desempeño de la localización local de robots. Se ha realizado un análisis de dichas estrategias "tradicionales"de los cuales se observó que tienen un coste computacional muy elevado en el caso de fusión sensorial, ya que requieren realizar la inversión y/o raíz cuadrada matricial de dimensiones altas para obtener la ganancia del filtro, lo que impide la implementación directa de este tipo de algoritmos en sistemas de recursos limitados y deben ser modificados para este fin.
- Se han seleccionado las estrategias ”modernas"de fusión sensorial más adecuadas como punto de partida para el presente trabajo, y se han realizado diversas reducciones y modificaciones sobre ellas para conseguir reducir sus costes temporales y computacionales consiguiendo su implementación en sistemas de recursos limitados.
- Distintas metodologías basadas en filtros de fusión sensorial como EKF, UKF y KF, se han adaptado, reducido e implementado en plataformas de recursos limitados para llevar a cabo una corrección continua de la localización local de robots en sistemas distribuidos, presentando resultados satisfactorios.
- Se ha definido las plataformas experimentales LEGO Mindstorms NXT, Summit XL y Rbcar, sobre las que se ha integrado los algoritmos de autolocalización local desarrollados.
- Finalmente, se han presentado pruebas experimentales de la implementación de los métodos de localización sobre los distintos tipos de robots con resultados satisfactorios que demuestran y validan las aportaciones de los algoritmos desarrollados en el presente capítulo.

Capítulo 4

Auto-localización Global para Interiores Basada en Correspondencias Geométricas

Las estrategias de localización global en interiores se basan principalmente en el reconocimiento del entorno del robot a través de la medida devuelta por algún sensor embarcado. Estos algoritmos tratan de resolver problemas como el del robot secuestrado (kidnapped robot [58]) o el del robot que despierta (wake-up robot [129]). En general existen dos metodologías para conseguirlo: la primera se basa en el estudio de algoritmos que tratan de auto-localizarse en un mapa y, al mismo tiempo, intentan construir el mapa completo del entorno, como el conocido algoritmo de SLAM [169]. Estas estrategias generalmente requieren un alto consumo de recursos, ya que utilizan sensores "avanzadosçomo cámaras RGB o sensores de barrido láser 3D, que requieren un procesamiento de imagen complejo y costosos algoritmos de correspondencia, por lo que son difícilmente aplicables a robots autónomos en tiempo real con recursos limitados. La segunda solución se basa en el refinamiento de la auto-localización a partir de un mapa conocido. El algoritmo de este tipo más popular es el de localización por Monte Carlo (MCL, [54]) el cual estima la posición del robot dentro del mapa conocido a medida que se mueve y va detectando su entorno. Típicamente, este algoritmo comienza con una distribución aleatoria uniforme de unas partículas cuya posición es candidata a coincidir con la posición estimada donde se encuentra el robot. Esta distribución se realiza sobre el mapa conocido del entorno y dichas partículas se vuelven a muestrear en base a una estimación bayesiana recursiva cuando el robot se mueve o cuando se detectan cambios en el escenario. Al mismo tiempo que el robot se desplaza por el entorno, este movimiento es aplicado sobre cada

partícula verificando posteriormente que la detección real obtenida por el robot es coherente según la nueva posición de cada partícula. La complejidad computacional del tiempo de aplicación del filtro de partículas depende del número de partículas y, naturalmente, cuantas más partículas, mayor es la precisión estimada, por lo que el algoritmo tiene un compromiso inherente entre velocidad y precisión. En los sistemas con recursos computacionales limitados no es viable analizar un gran número de partículas cada vez, por lo que generalmente, o la precisión es pobre, o se hace uso de comunicaciones para realizar los cálculos en un ordenador remoto con mayores recursos computacionales.

En este capítulo se describe un algoritmo de localización global aplicable a robots con recursos limitados que parte de la misma idea que el algoritmo Monte Carlo (el conocimiento previo de un mapa conocido), pero no hace uso de sistemas de partículas sino que utiliza estrategias de correspondencia geométrica procesadas de manera local, dentro del robot.

4.1 Algoritmo de Localización Global Geométrica

Las técnicas de auto-localización de robots basadas en el conocimiento previo de un mapa conocido, se centran principalmente en el cálculo de la correspondencia de las mediciones u observaciones del entorno. Estos métodos pueden ser entendidos como una aplicación específica de las técnicas de registro de superficies donde lo que se pretende es calcular la mejor correspondencia entre dos muestras. Desde el punto de vista de la robótica, estas dos muestras podrían consistir en la medida observada desde la posición actual del robot y la medida observada desde una posición anterior. Conociendo el desplazamiento relativo del robot entre las dos muestras, es posible obtener su posición mediante una simple triangulación. En esta línea de investigación, se han propuesto múltiples descriptores que simplifican el proceso de búsqueda basado en niveles de curvatura como [69] y [103], o invariantes integrales como [81] y [71]. También se han propuesto técnicas para calcular la alineación del descriptor basadas en optimización combinatoria ([71] y [155]), algoritmos aleatorios RANSAC como [103] y [160] o métodos de búsqueda hacia adelante como [69]. Los resultados obtenidos con estas técnicas proporcionan alineamientos burdos que generalmente se refinan con algoritmos como ICP (Iterative Closest Point, [21],[41]), el cual alterna el cálculo de la correspondencia entre las muestras y el cálculo de la transformación de la alineación de las mismas. Existen muchas variaciones del algoritmo original que se pueden encontrar adecuadamente clasificadas y detalladas en [152].

Teniendo en cuenta estos factores, se ha desarrollado GEMA^2(Geometrical Matching Analytical Algorithm), un algoritmo analítico para la auto-localización de robots basado en la correspondencia geométrica de las muestras 2D del entorno recibidas mediante un sensor láser, con el fin de proporcionar un algoritmo efi-

ciente de localización global compatible con estrategias de localización local y computacionalmente implementable en sistemas con recursos limitados.

La técnica propuesta se aprovecha de la naturaleza estructurada de los ambientes en interiores de edificios o construcciones. Esta estructura se compone en la mayoría de casos, de la presencia de paredes rectas que forman muros o pilares, por lo que el algoritmo centra su esfuerzo en el desarrollo de un método rápido de extracción de líneas con el fin de identificar las esquinas características del entorno para localizar la posición del robot. Sin embargo, el algoritmo propuesto puede ser fácilmente extendido para reconocer cualquier otro tipo de primitivas geométricas parametrizables usando la misma idea.

4.2 Metodología del Algoritmo GEMA^2

El algoritmo de auto-localización desarrollado GEMA^2 se puede dividir en 3 fases principalmente: calibración, segmentación y localización. A su vez, las fases de segmentación y localización se han dividido en más subtareas con el fin de descomponer mejor las funcionalidades que incluye cada una.

La etapa de *Calibración* se ejecuta de manera previa, aislada e independiente al algoritmo, es decir, antes de que el robot comience su misión o empiece a moverse de manera autónoma. Además, es posible utilizar los resultados de esta etapa en las etapas siguientes sin necesidad de volver a calibrar el sensor. El objetivo de esta etapa es calcular la función de error $e(d)$ asociado a las mediciones de las distancias d_i devueltas por el sensor.

Una vez que se ha llevado a cabo la fase de *Calibración*, el ciclo de ejecución del algoritmo comienza leyendo las medidas obtenidas por el sensor láser y transformando el conjunto de puntos obtenidos 2D locales a una representación más sencilla de procesar y analizar. Esta tarea se realiza en la fase de *Segmentación*, donde los puntos s_i se infieren en lineas l_i, las cuales se filtran para posteriormente calcular sus respectivas intersecciones con el objetivo de obtener el conjunto de esquinas detectadas c_i.

La esquina mejor definida c entra en la fase de *Localización* donde se alinea con todas las esquinas similares del mapa conocido m_i. Cada alineación supone una posible localización del robot (θ_{robot}, x_{robot}, y_{robot}), la cual se evalúa utilizando una función de coste basada en errores cuadráticos. La alineación cuyo error es el más pequeño es devuelta como la ubicación resultante del robot dentro del mapa conocido.

La siguiente figura 4.1 muestra un esquema de las fases que componen el algoritmo donde se ha especificado las entradas y las salidas de cada fase y subtarea.

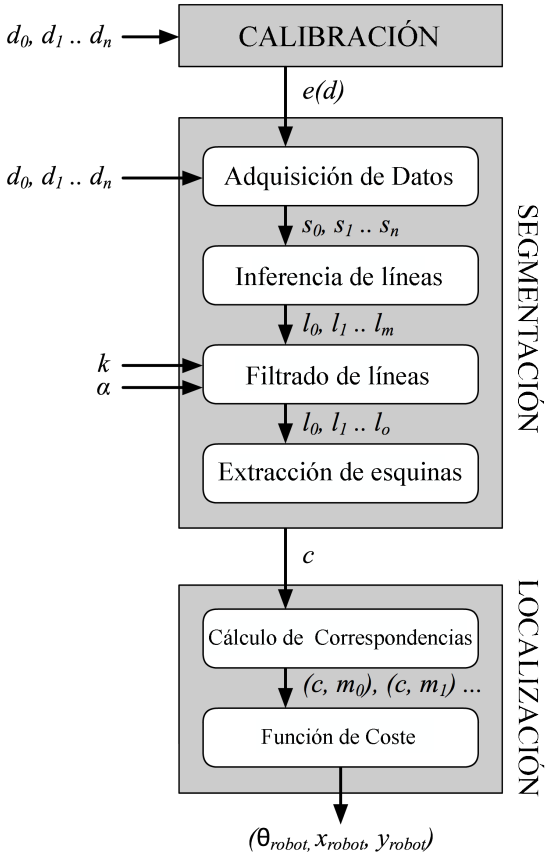


Figura 4.1: Esquema del ciclo de ejecución del algoritmo GEMA^2.

4.2.1 Fase de Calibración

La fase de calibración estima el error de la medida del sensor láser utilizado. El resultado de esta etapa es crítico ya que el algoritmo de inferencia de líneas hace uso de este valor para establecer si una muestra dada pertenece a la misma línea definida por muestras anteriores o si, por el contrario, forma parte de una línea nueva.

El error de medida $e(d)$ es una función que indica, para cada distancia d_i, la incertidumbre del valor devuelto. Su valor puede ser una constante o una función que dependa de la distancia y/o la orientación del haz del láser, medido en referencia a las coordenadas locales del robot.

Para llevar a cabo la calibración del sensor, el robot (O) se coloca en frente de una pared recta y, mediante el método de mínimos cuadrados se calcula la ecuación de la recta (l) que mejor se adapte a las mediciones registradas. El error asociado a cada distancia ($e(d_n)$) se calcula como la distancia geométrica entre el punto devuelto por el sensor (s_n) y, el punto definido por la intersección entre su línea de proyección y la línea óptima calculada utilizando todas las muestras, es decir, $\overline{Os_n} \cap l$. La figura 4.2 ilustra este concepto.

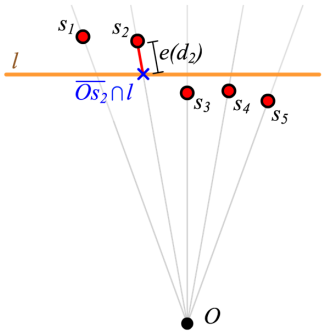


Figura 4.2: Cálculo del error asociado a la medida s_2. En naranja se muestra la línea calculada mediante mínimos cuadrados.

Durante los experimentos empíricos realizados en este trabajo, el error de las medidas del láser ha permanecido constante independientemente de la distancia obtenida o el ángulo de proyección de las mismas, de modo que se ha establecido la siguiente función de error a modo de un valor constante para todas las medidas: $e(d)= 2cm$. Valor que se corresponde con la cota máxima del error obtenido en esta etapa de calibración.

4.2.2 Fase de Segmentación

La finalidad de la fase de segmentación es reducir el conjunto de datos de entrada a una estructura de datos simplificada que permita compararlos con la representación del mapa conocido de una manera eficiente y robusta. Para ello, el ruido introducido por el sensor y las imperfecciones de los objetos del mundo real deben ser filtrados utilizando la estimación del error calculado en la etapa de calibración.

Para acelerar los cálculos en la fase de localización, el conjunto de medidas de distancias obtenidas mediante el láser se infiere en un conjunto de líneas rectas donde quedan comprendidas dichas medidas. Estas líneas se intersecan con las líneas vecinas con el objetivo de encontrar esquinas características. La esquina que se estime más precisa será la utilizada en la etapa de localización para evaluar la posición del robot dentro del mapa conocido. Todo este proceso se divide en subtareas, las cuales se detallan a continuación.

4.2.2.1 Adquisición de datos

El sensor láser se define como $S(f, n, e(d))$ donde f representa el campo de vista o FOV (Field Of View), n es el número de muestras recogidas y $e(d)$ es la función de error que estima la medida del error asociado a cada distancia d_i.

Los valores de f y n son especificados por el fabricante del sensor mientras que el valor $e(d)$ se calcula en la etapa de calibración.

El conjunto de medidas de distancia obtenidas por el sensor $D = (d_1, d_2, ..., d_n)$ se corresponden con un vector ordenado de valores decimales. El ángulo de proyección φ_i de una distancia d_i se puede calcular como $\varphi_i = f \cdot (\frac{i}{n} - \frac{1}{2})$, y sus coordenadas locales (u_i,v_i) como sigue:

$$u_i = \sin(\varphi_i) \cdot d_i \tag{4.1}$$

$$v_i = \cos(\varphi_i) \cdot d_i \tag{4.2}$$

Una vez obtenido el conjunto de distancias D, los valores de las medidas son convertidos al formato $S = (s_1, s_2, ...s_n)$ donde cada muestra s_i se define mediante tres puntos referenciados respecto al sistema de coordenadas local del sensor láser.

$$s_i = (\sin(\varphi_i) \cdot d_i, \cos(\varphi_i) \cdot d_i) \tag{4.3}$$

$$\lfloor s_i \rfloor = (\sin(\varphi_i) \cdot (d_i - |e(d_i)|), \cos(\varphi_i) \cdot (d_i - |e(d_i)|)) \tag{4.4}$$

$$\lceil s_i \rceil = (\sin(\varphi_i) \cdot (d_i + |e(d_i)|), \cos(\varphi_i) \cdot (d_i + |e(d_i)|)) \tag{4.5}$$

donde $e(d_i)$ es el error de medida asociado a cada muestra.

De este modo, cada muestra queda caracterizada por tres valores: la distancia medida por el sensor s_i, su distancia mínima posible $\lfloor s_i \rfloor$, y su distancia máxima posible $\lceil s_i \rceil$, según la estimación del error de medida calculado en la etapa de calibración.

Estas muestras se utilizan en las siguientes fases del algoritmo para llevar a cabo distintos cálculos. Si la frecuencia requerida por la técnica propuesta no pudiera ser efectiva en términos de capacidad computacional debido a restricciones de cálculo del sistema, siempre sería posible reducir el número de muestras de modo que cada muestra calculada represente un conjunto de medidas de distancia. Así pues, mediante el promedio de las distancias, la cantidad de muestras a procesar en las etapas posteriores se reduciría y al mismo tiempo se filtraría algo de ruido aleatorio proveniente de las muestras.

4.2.2.2 *Inferencia de líneas*

El objetivo de esta fase es calcular el conjunto de líneas rectas $L = (l_1, l_2, ...l_m)$ que mejor se ajuste a las muestras calculadas durante la etapa de adquisición del algoritmo, teniendo en cuenta el error de medición de cada una.

El método de inferencia de líneas itera las muestras en orden, creando nuevas líneas cuando es necesario. La principal dificultad asociada a estos cálculos reside en establecer si una muestra dada s_i pertenece a la línea obtenida mediante las muestras anteriores s_{i-j}, o si forma parte de una línea nueva.

Para resolver este problema, se ha asociado a cada línea l_k un estimador llamado *continuidad de la línea*, V_{l_k}, el cual se ha definido como $V_{l_k} = (d_{min}, d_{max})$. Estos valores d_{min} y d_{max} especifican el rango de la distancia proyectada en cada muestra s_i dentro del cual se considera que la muestra i forma parte de la línea definida mediante muestras anteriores. De este modo, se establece que:

$$s_i \in l_k \leftrightarrow [\lfloor s_i \rfloor ... \lceil s_i \rceil] \cap [d_{min} ... d_{max}] \neq \emptyset \tag{4.6}$$

siendo d_{min} y d_{max} el rango de distancias especificado por V_{l_k}, y l_k la línea definida mediante las muestras anteriores $(s_{i-1}, s_{i-2}, ...)$. Los valores $[\lfloor s_i \rfloor ... \lceil s_i \rceil]$ representan la incertidumbre asociada a la posición de s_i teniendo en cuenta su error asociado.

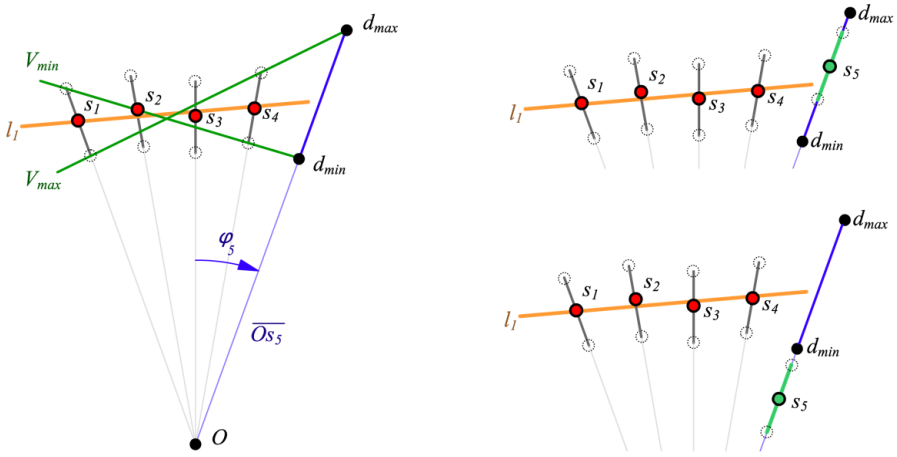


Figura 4.3: Funcionamiento del estimador implementado: *continuidad de la línea.*

La figura 4.3 muestra visualmente este concepto. Los círculos rojos corresponden a los valores s_i y los círculos discontinuos corresponden a los valores $\lfloor s_i \rfloor$ y $\lceil s_i \rceil$.

La figura de la izquierda muestra la representación del concepto de continuidad V_{l_1} sobre la línea l_1 teniendo en cuenta el conjunto de las cuatro muestras ([s_1, s_2, s_3, s_4]) procesadas hasta el momento. Para estudiar una nueva muestra s_5, es necesario encontrar las pendientes máximas y mínimas de las rectas, V_{max} y V_{min}, que intersecan con todos los rangos calculados de $[\lfloor s_k \rfloor ... \lceil s_k \rceil]$, donde k se corresponde con el número de muestras actuales que forman parte de l_k. La intersección entre estas líneas V_{max} y V_{min} con la línea $\overline{Os_l}$ definida por φ_i y representada en azul en la figura 4.3, corresponde a los puntos calculados d_{min} y d_{max}, siendo $d_{min} = V_{min} \cap \overline{Os_l}$ y $d_{max} = V_{max} \cap \overline{Os_l}$.

De esta manera, si una muestra S_i, cumple las ecuaciones 4.1 y 4.2, se incluirá como parte de la línea definida, y los valores de d_{min} y d_{max} serán re-calculados para la siguiente muestra. El dibujo superior derecho de la figura 4.3 muestra este caso para la muestra s_5 representada en color verde.

Por el contrario, en el dibujo inferior derecho se representa la muestra s_5 de forma que no cumple las condiciones de las ecuaciones 4.1 y 4.2, por lo que no existe una línea recta que interseque con todos los rangos de $[\lfloor s_k \rfloor ... \lceil s_k \rceil], s_k \in (S \cup s_i)$. En tal caso, dicha muestra formará parte de una nueva línea distinta a l_1 y la ecuación de la recta l_1 será calculada mediante mínimos cuadrados de acuerdo con la distancia de proyección medida para cada muestra s_i incluida en l_1.

Algoritmo 8: Algoritmo de clustering desarrollado para la etapa de inferencias en líneas. Devuelve el conjunto de líneas L que mejor se corresponde con el número de muestras S

Entrada: $S = [\ s_0, s_1, ..., s_n\ ]$, $D = [\ d_1, d_2, ..., d_n\ ]$
Salida: $L = [\ l_0, l_1, ..., l_m\ ]$
Inicialización:
    $L \leftarrow \{\emptyset\}$;
    $l \leftarrow$ Nueva_Línea($\emptyset$);
Para cada *muestra* s_i *en* S **hacer**
    $d_{min} \leftarrow V_l.min$;
    $d_{max} \leftarrow V_l.max$;
    Si ($[\lfloor s_i \rfloor ... \lceil s_i \rceil)] \cap [d_{min} ... d_{max}] \neq \emptyset$) **entonces**
        l.AñadirMuestra(s_i);
        l.RecalcularV_l();
    si no
        l.AjustePorMinimosCuadrados();
        L.add(l);
        l $\leftarrow$ nueva_línea(s_i);
    fin
fin
l.AjustePorMinimosCuadrados();
L.añadir(l);

El algoritmo 8 muestra en pseudocódigo el método de clustering desarrollado para esta etapa. Sintetizando, el algoritmo organiza de manera ordenada las muestras originales obtenidas mediante el láser en un conjunto L de líneas rectas formadas por subconjuntos de muestras alineadas.

Una de las ventajas más destacable de esta técnica es que, primero produce el conjunto de muestras alineadas y, a continuación se calcula la ecuación de la línea óptima que mejor se adapta a dicho conjunto. Este orden en la ejecución de los cálculos es importante ya que el método de mínimos cuadrados ordinarios (Ordinary Least Squares or Linear Least Squares) minimiza la suma del cuadrado de las distancias verticales entre muestras y las respuestas predichas por aproximación lineal. Por tanto, cuanto más horizontalmente se distribuyen las muestras detectadas, mejor precisión ofrece el algoritmo. Para aprovechar este hecho y con el fin de optimizar los resultados de esta técnica, el conjunto de muestras de un grupo puede ser rotado inicialmente hasta alcanzar una orientación horizontal, para así aplicar el algoritmo de manera óptima y luego, a posteriori, volverlo a rotar hasta su posición original.

Como se ha comentado anteriormente, el cálculo de los valores d_{min} y d_{max} depende directamente de las pendientes mínima y máxima de las rectas V_{min} y V_{max}. Teniendo en cuenta que la manera de calcular estas líneas es simétrica, mediante uno de los dos valores es posible obtener el otro directamente. Dada una muestra nueva s_i, el algoritmo desarrollado se centra en el cálculo de V_{max} considerando dos casos diferentes:

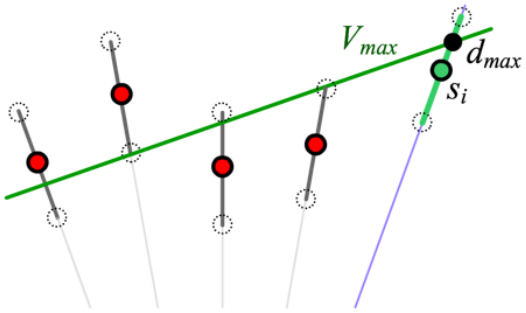


Figura 4.4: Caso A. Se cumple que $\lfloor\rfloor s_i \lfloor ... \rceil s_i \lceil\rceil$ interseca con V_{max} por lo que no es necesario llevar a cabo la actualización.

A. Si $\lfloor\rfloor s_i \lfloor ... \rceil s_i \lceil\rceil$ interseca con V_{max}, en este caso no es necesario actualizar, ya que V_{max} ya define la pendiente máxima que incluye la nueva muestra. Este caso se muestra en la figura 4.4.

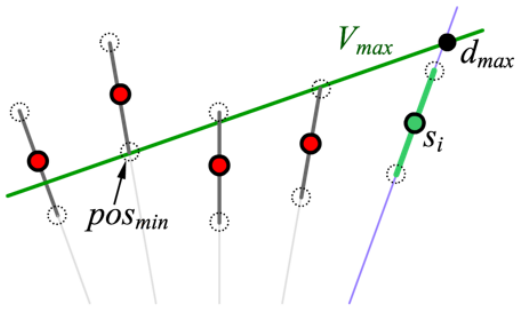


Figura 4.5: Caso B, donde se requiere actualizar la pendiente de V_{max} para que incluya la nueva muestra S_i.

B. Si ($\lceil s_i \rceil \geq d_{max}$), quiere decir que teniendo en cuenta la muestra nueva s_i, V_{max} ya no contiene la pendiente máxima tal y conforme ilustra la figura 4.5 y tiene que ser re-calculada.

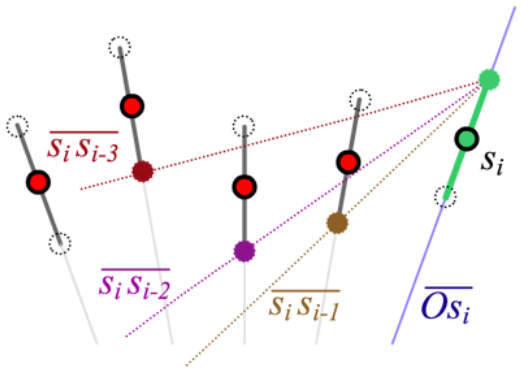


Figura 4.6: Conjunto de pendientes candidatas a ser la nueva recta V_{max} re-calculada. En el ejemplo que se muestra, la recta V_{max} será finalmente la recta definida por $\overline{\lceil s_i \rceil \lfloor s_{i-3} \rfloor}$

Para re-calcular una nueva pendiente de V_{max} teniendo en cuenta la nueva muestra s_i, hay que tener en cuenta la distancia proyectada más restrictiva de todos los puntos. Suponiendo que la nueva línea V_{max} va a estar definida por los valores $\lfloor s_q \rfloor$ y $\lceil s_q \rceil$, la correspondencia de $\lceil s_q \rceil$ con s_i resulta inmediata dado que $\lceil s_q \rceil < d_{max}$. En cambio, para calcular el valor de $\lfloor s_q \rfloor$ es necesario iterar a través de cada muestra incluida en la línea actual con el fin de buscar el punto $\lfloor s_j \rfloor$ que minimice el ángulo entre la línea $\overline{\lceil s_i \rceil \lfloor s_j \rfloor}$ y la línea proyectada definida por $\overline{Os_i}$. En la figura 4.6 puede observarse las distintas líneas candidatas a ser V_{max}.

Teniendo en cuenta que la adición de muestras a una línea es un proceso iterativo, una optimización interesante aplicada al algoritmo es que, una vez que una muestra ha sido identificada como la más restrictiva de todas, ninguna de las muestras anteriores debe ser considerada en iteraciones posteriores. De esta manera, almacenando el valor del índice de la muestra más restrictiva se eliminan una gran cantidad de evaluaciones innecesarias, asegurando al mismo tiempo un resultado correcto.

El código en pseudocódigo de la función encargada de re-calcular el valor de V_{max} cuando es necesario, se muestra en el algoritmo 9.

Algoritmo 9: Pseudo-código para actualizar el valor de la recta V_{max}. Considerando que pos_{min} es una variable global que guarda su valor después de cada invocación a la función.

```
function ActualizarV_max (S_i: Nueva_Muestra, V_max: Línea)
    d_max ← V_max ∩ Os_i ;
    Si ⌈s_i⌉)<d_max entonces
        Angulo_min ← ∞ ;
        Para cada pos en pos_min hasta i − 1 hacer
            Si Ángulo(⌈s_i⌉⌊s_pos⌋, Os_i) <angulo_min entonces
                angulo_min ← Angulo(⌈s_i⌉⌊s_pos⌋, Os_i);
                posmin ← pos;
            fin
        fin
        V_max ← ⌈s_i⌉⌊s_pos_min⌋
    fin
return V_max
```

En términos de complejidad computacional, el peor caso que puede ocurrir en el algoritmo descrito es que las muestras obtenidas sean un conjunto de muestras perfectamente alineadas. Todas ellas cumplirían la restricción del estimador de *continuidad de la línea* al formar una única línea. De este modo, para cada muestra nueva se actualizaría necesariamente V_{min} y V_{max} teniendo en cuenta todas las muestras anteriores, a pesar de que la más restrictiva siempre sería la primera. Por lo que el coste de ejecución sería de $O(n^2)$, siendo n el número de muestras. Sin embargo, este caso difícilmente ocurre en casos donde el FOV del sensor es igual o mayor a 180º.

El mejor de los casos que puede ocurrir, es que cada muestra nueva forme parte de una línea nueva. En este caso nunca sería necesaria la actualización de V_{min} y V_{max}, y el coste de ejecución del algoritmo sería $O(n)$.

4.2.2.3 Filtrado de líneas

El objetivo de esta etapa es añadir robustez al algoritmo descrito considerando nuevas incertidumbres no contempladas cuando se aplica la función de error a las medidas del sensor $e(d)$. Estas incertidumbres principalmente suelen provenir de imperfecciones físicas tanto del sensor como del entorno.

Durante las pruebas empíricas, dichas imperfecciones han sido clasificadas en dos grupos: (a) interferencias debidas a la topología del material del entorno contra el que chocan los haces de luz del láser, y (b), las imperfecciones físicas en las superficies de las paredes o pilares.

Las interferencias en el escáner láser (caso (a)), suelen ocurrir cuando los haces de luz rebotan contra superficies reflexivas o refractivas. En estos casos, las medidas recibidas como datos de entrada del algoritmo pueden ser impredecibles, pudiendo contener un alto nivel de ruido.

Por otro lado, las imperfecciones en las superficies de los muros u objetos (caso (b)) pueden afectar a la técnica de inferencia de rectas resultando que para una misma pared, el algoritmo la divida en líneas rectas distintas con motivo de que el estimador de *continuidad de línea* falle debido al ruido producido por dichas imperfecciones.

Mediante la detección y el apropiado filtrado de estas imperfecciones la calidad de los resultados mejora significativamente, y esta es la finalidad de la presente fase.

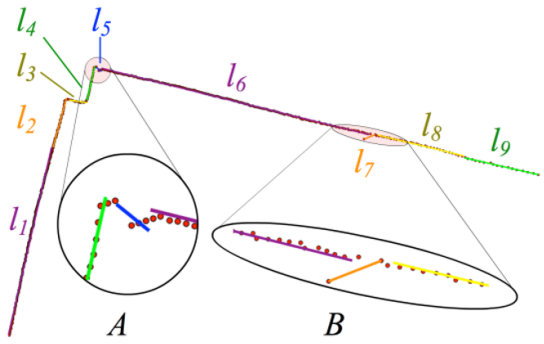


Figura 4.7: Resultado de la inferencia de líneas previa a la aplicación de la presente etapa de filtrado, con imperfecciones reflejadas en las medidas capturadas por el sensor láser.

Para filtrar las interferencias del escáner hay que considerar el hecho de que una medida anómala penaliza seriamente el cálculo del estimador de *continuidad de la línea* y por tanto suele generar más líneas de las necesarias donde cada línea tiene

un número muy reducido de medidas. En los casos A y B de la imagen 4.7 se puede observar este hecho donde existen líneas nuevas inferidas por tan solo dos o tres medidas. Por tanto, todas las líneas que contengan menos de k muestras pueden ser descartadas junto con sus muestras si $|l_i| < k$, siendo el valor de k un parámetro especificado por el usuario. Las pruebas empíricas con el hardware descrito en la sección 4.3 han demostrado que un rango de valores entre [4 ... 6] para k puede ser el adecuado para obtener unos buenos resultados.

La imagen 4.7 muestra los resultados de la inferencia de líneas sin aplicar el filtrado correspondiente. Se puede intuir una estructura formada por dos paredes que convergen en un pilar sobresaliente en una esquina, lo cual debería inferirse en 4 líneas rectas y sin embargo el algoritmo infiere un total de 9 líneas. Las muestras que penalizan al algoritmo se han resaltado en los casos A y B. Aplicando el filtrado con $k = 6$, las líneas l_5 y l_7 son descartadas por tener un número de muestras menor que k. El caso de la división entre l_1 y l_2 es debido a las interferencias del escáner y para evitarlo se aplica un segundo filtro con un parámetro α, que indica la máxima tolerancia de la desviación de los muros. Todas las líneas consecutivas con una desviación menor que α, son combinadas y re-calculadas considerando todas sus muestras. Este segundo filtrado unifica las líneas l_1 y l_2, obteniendo así como resultado final la inferencia en 4 líneas que se buscaba tal y conforme muestra la figura 4.8.

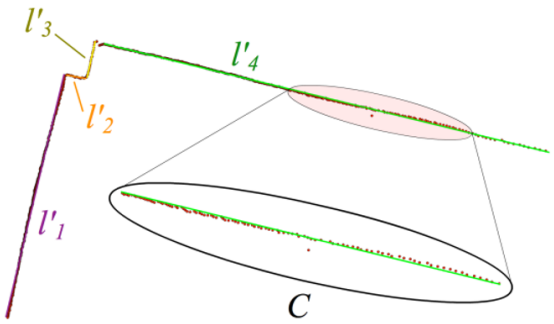


Figura 4.8: Resultado de la inferencia de líneas después de haber aplicado los dos métodos de filtrado.

El resultado devuelto de esta etapa es un conjunto de líneas $L' = (l'_1, l'_2, ...l'_o)$ que mejora la segmentación obtenida en la etapa anterior $L = (l_1, l_2, ...l_m)$, y verifica que $o \leq m$.

4.2.2.4 Extracción de esquinas

Para comparar eficientemente los resultados del conjunto de líneas L' con la representación del mapa conocido, es necesario llevar a cabo una correspondencia global entre los dos conjuntos de datos. Una de las técnicas más comunes para esto es intentar simplificar al máximo la representación de ambos conjuntos de manera que los cálculos de correspondencia se aceleran considerablemente.

En lugar de trabajar directamente con líneas, esta etapa utiliza de esquinas para llevar a cabo las correspondencias. Una esquina se define como la intersección de dos líneas consecutivas, y se caracteriza por su ángulo (β), su posición (x, y) y su orientación (θ).

Para obtener el conjunto de esquinas $C = (c_1, c_2, ...c_p)$ definidas por $L' = (l'_1, l'_2, ...l'_o)$, todas las líneas se intersecan entre sí almacenando tan sólo las intersecciones más cercanas a los finales de ambas líneas. Mediante esta estrategia, todas las esquinas que se calculan puede que no existan en el mundo real, pero resultan muy útiles para el algoritmo de localización el cual se explica en la siguiente sección.

Dado el interés por acelerar los cálculos de correspondencia, en lugar de utilizar todas las esquinas calculadas en esta fase, sólo se utiliza la mejor caracterizada c. Para encontrarla dentro del conjunto C, la calidad de cada esquina se define como $|c_i| = min(|l_1|, |l_2|)$, siendo l_1 y l_2 las líneas que han generado dicha esquina, y $|l_1|$ y $|l_2|$ el conjunto de cada línea respectivamente. De manera formal:

$$\begin{aligned} c &= max(|c_i|), c_i \in C \\ |c_i| &= min(|l_1|, |l_2|), l_1 \in c_i, l_2 \in c_i \end{aligned} \tag{4.7}$$

4.2.3 Fase de Localización

Los datos de entrada de la presente fase se corresponden con el conjunto de esquinas extraídas en la etapa de segmentación y el conjunto de esquinas extraídas del mapa conocido. Mediante la alineación de estas esquinas se obtienen todas las posibles ubicaciones del robot.

Para cada posición candidata, se calcula una medida de corrección para ordenarlos de acuerdo a la calidad de los resultados obtenidos, esperando que el mejor resultado se ajuste a la localización del robot en el mundo real.

4.2.3.1 Cálculo de Correspondencias

Esta etapa se encarga de realizar el cálculo de todas las posibles ubicaciones del robot dentro del mapa conocido. Para ello se alinean todas las esquinas del mapa conocido que poseen un ángulo interno similar al obtenido de los datos de salida de la etapa de segmentación, lo cual produce una localización candidata por cada alineación.

Con el fin de acelerar este cálculo, el mapa conocido se pre-procesa para obtener todas las esquinas ordenadas en función de la medida de su ángulo interno. De este modo, el coste computacional de la búsqueda de ángulos similares se reduce a $O(log_2(n))$, siendo n el número total de esquinas del mapa. En caso de que el mapa sea demasiado grande, después de una primera aproximación a una ubicación correcta, tan sólo es necesario comprobar las esquinas de alrededor de esa última posición conocida del robot, para evitar cálculos innecesarios.

Dada una correspondencia entre esquinas, la orientación del robot (θ_{robot}) se calcula como sigue:

$$\theta_{robot} = \frac{\pi}{2} + \theta_m - \theta_c - \frac{(\beta_m - \beta_c) \cdot |l_{1_c}|}{|l_{1_c}| + |l_{2_c}|} \tag{4.8}$$

Donde θ_m es la orientación de la esquina seleccionada del mapa (en coordenadas globales), θ_c es la orientación de la esquina calculada en la fase de segmentación (en coordenadas locales), β_m es el ángulo interno de la esquina seleccionada en el mapa, β_c es el ángulo interno de la esquina calculada en la fase de segmentación, y $|l_{1_c}$, $|l_{2_c}$ son los conjuntos de muestras contenidas en las líneas que definen la esquina calculada en la anterior etapa.

Mediante el cálculo de θ_{robot} según la ecuación 4.8, la diferencia entre los ángulos internos se compensa proporcionalmente según la cantidad de muestras de cada línea que define la esquina c. Cuantas más muestras forman una línea, más precisa es la estimación.

La posición del robot (x_{robot},y_{robot}) se infiere mediante triangulación como a continuación muestra la siguiente ecuación:

$$\begin{aligned} x_{robot} &= x_c \cdot \cos(\theta_{robot}) + y_c \cdot \sin(\theta_{robot}) + x_m \\ y_{robot} &= x_c \cdot \sin(\theta_{robot}) - y_c \cdot \cos(\theta_{robot}) + y_m \end{aligned} \tag{4.9}$$

Donde x_m y y_m son las coordenadas globales del punto de intersección definido por la esquina seleccionada en el mapa, y x_c y y_c son las coordenadas locales del punto de intersección definido por la esquina calculada en la fase de segmentación.

La figura 4.9 muestra visualmente la alineación entre la esquina c y una esquina procedente del mapa conocido.

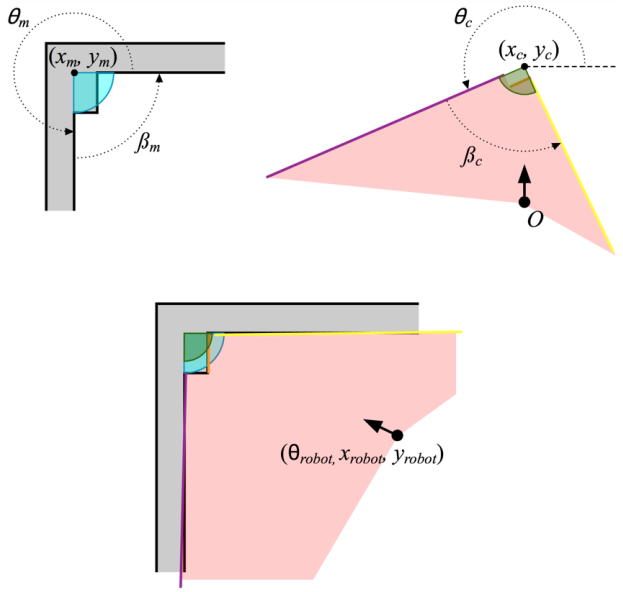


Figura 4.9: Representación visual de la técnica de correspondencia entre esquinas.

La primera imagen (arriba-izquierda) muestra la caracterización de una esquina del mapa conocido en coordenadas globales, su posición x_m, y_m, su ángulo interno β_m y su externo θ_m. La segunda imagen (arriba-derecha) muestra la caracterización de la esquina seleccionada del conjunto de muestras de entrada c en coordenadas locales y obtenida desde la posición O del láser. La imagen inferior muestra la alineación correspondiente entre las dos esquinas mediante el ángulo interno y la posición final del robot en coordenadas globales $(\theta_{robot}, x_{robot}, y_{robot})$.

4.2.3.2 Función de coste

Esta última etapa del algoritmo tiene como finalidad evaluar la calidad de cada posición candidata devuelta por la etapa anterior. El resultado final del algoritmo de localización es una lista ordenada de posibles posiciones donde se espera que el primer elemento se corresponda con la posición del robot en el mundo real.

Para cuantificar la calidad de cada posición se utiliza el mapa conocido. Se calcula el conjunto de medidas $D' = (d'_1, d'_2, ...d'_n)$ que el sensor debería haber producido en condiciones ideales (sin errores de medida) si el robot se posicionara en la ubicación calculada.

El error de orientación total, puede calcularse como:

$$error = \sum_{0 \leq i \leq n} (d_i - d'_i)^2 \tag{4.10}$$

donde d_i son las medidas originales devueltas por el sensor y n el número de medidas.

En caso de que el coste computacional de estos cálculos no satisficiera la frecuencia requerida por el algoritmo de localización global, como ocurre con robots de recursos computacionales limitados, se puede utilizar tan sólo un subconjunto equidistante de las muestras de distancias.

4.3 Plataforma Experimental: Robotino

Con el objetivo de verificar el correcto funcionamiento del algoritmo GEMA^2, se ha llevado a cabo su implementación en la plataforma experimental de recursos limitados *Robotino*® de la empresa FESTO [23] descrita a continuación.

Robotino es un robot móvil formado por tres ruedas omnidireccionales el cual posee un computador empotrado con el sistema operativo Linux instalado en una tarjeta compact flash.

Con el objetivo de mejorar su funcionalidad, Robotino se ha equipado con distintos tipos de sensores como encoders incrementales instalados en las ruedas, sensores de contacto para detección de colisiones, sensores de proximidad mediante infrarrojos, un brazo manipulador y, en concreto para este trabajo, un sensor láser 2D.

Robotino (figura 4.10) es accionado por 3 ruedas omnidireccionales independientes montadas en un ángulo de 120º entre sí. Esto permite al robot poder moverse en todas las direcciones sin necesidad de modificar el ángulo de su eje de rotación. Cada una de las 3 unidades de accionamiento consta de un motor de corriente continua, un engranaje, un rodillo, una correa dentada y un encoder incremental que permite regular cada rueda mediante un controlador PID.

La unidad de control consta de tres componentes:

- Procesador PC104, compatible con MOPSIcdVE a 300 MHz con un sistema operativo Linux con un kernel en tiempo real, memoria SDRAM de 128 MB, conexión Ethernet, USB e interfaz VGA.
- Una tarjeta compact flash con el sistema operativo y el FRS ROS.
- Un punto de acceso inalámbrico.

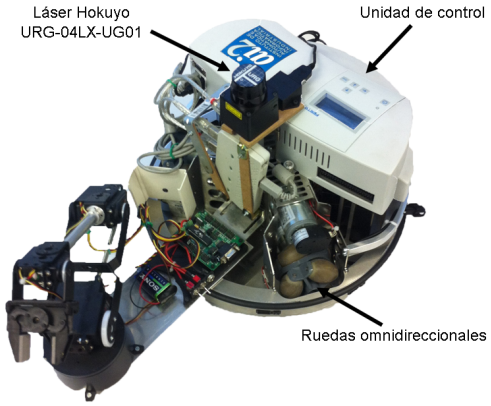


Figura 4.10: Robot móvil Robotino de configuración omnidireccional con Hokuyo láser instalado.

Para la detección del entorno, se ha instalado en el robot un láser Hokuyo URG-04LX-UG01 [124]. Este láser 2D emite una luz infrarroja sobre un área semicircular de 240º con un radio máximo de 5600 mm. El sensor tiene una resolución angular de 0.36º devolviendo como máximo un total de 684 medidas. Su principio de medición de la distancia se basa en el cálculo del desfase del envío y la recepción de la luz lo que le permite obtener mediciones estables con influencias mínimas del color y de la reflectancia del objeto.

El sensor láser Hokuyo se conecta vía USB a Robotino. El framework ROS ofrece soporte directo para este tipo de sensores mediante el lanzamiento del nodo *hokuyo_laser* el cual principalmente se encarga de publicar las medidas recogidas por el láser a través de un tópic.

4.4 Pruebas Experimentales

El algoritmo $GEMA^2$ se ha implementado como un nodo independiente de ROS en lenguaje C++. Este nodo se suscribe al tópic del nodo *hokuyo_laser* que ofrece las 684 medidas del láser de manera asíncrona con un periodo de 400 ms. La etapa de calibración del algoritmo se ha implementado como un servicio ofrecido por el nodo, de modo que es posible recalibrar dicha etapa mediante una simple llamada a dicho servicio.

Aprovechando las ventajas que ofrece ROS, la posición resultante sugerida por el algoritmo $GEMA^2$ se publica a través de otro tópic independiente, de modo que cualquier nodo interesado puede acceder a dicha información mediante una

simple suscripción. Esto resulta muy útil para nodos de fusión sensorial como el *robot_localization* o el *robot_pose_ekf*.Estos paquetes estiman la posición del robot a partir de distintas fuentes de información relacionadas con su localización local y global. La posición sugerida por $GEMA^2$ se introduce como una estimación de posición más junto a otras como la estimación local a partir de encoders o de sensores de movimiento como unidades inerciales, acelerómetros, giróscopos, etc. y mediante la fusión de dichos datos de entrada, se propone finalmente una posición del robot con mejor estimación. Además, la modularidad de ROS permite que el algoritmo $GEMA^2$ pueda ser encapsulado dentro de un paquete independiente y utilizado bajo cualquier sistema que funcione con ROS. En el código QR de la derecha o a través del siguiente enlace se puede visualizar el correcto funcionamiento de $GEMA^2$ implementado en el robot Robotino:`/https://www.youtube.com/watch?v=2nFdBIAUEsw`

4.5 Resultados Obtenidos en Pruebas Empíricas

Se han llevado a cabo multitud de pruebas con el robot Robotino y el láser Hokuyo embarcado para calcular el coste computacional del algoritmo en función del número de muestras recogidas por el láser. La figura 4.11 muestra la relación entre el tiempo de cómputo necesario para ejecutar $GEMA^2$ dentro del robot móvil Robotino y su relación con el número de muestras procesadas.

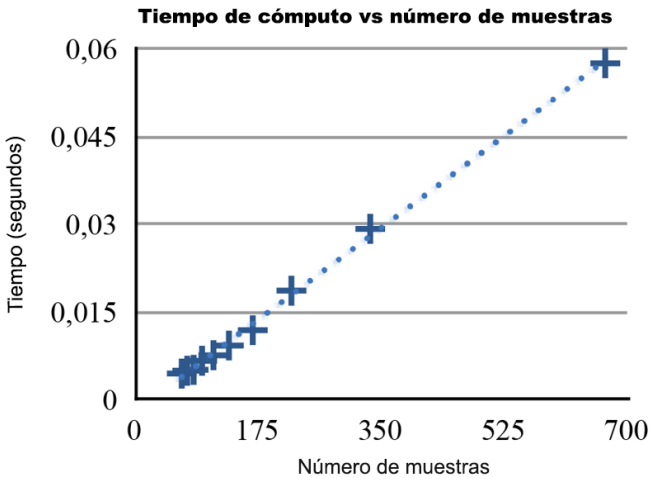


Figura 4.11: Relación entre el tiempo necesario de ejecución del algoritmo $GEMA^2$ en Robotino en relación con el número de muestras recibidas por el láser Hokuyo.

Como puede observarse en la figura anterior, el coste computacional del algoritmo es lineal respecto al número de muestras. El peor caso ocurre cuando se utilizan todas las muestras recogidas por el láser (684 muestras). Para tal caso, $GEMA^2$ requiere un tiempo de cómputo de 0,058 segundos, lo cual no supone ningún problema para el procesador de Robotino y además permite un alto periodo de publicación del resultado de la localización a través del tópic de salida de ROS.

Para cuantificar la calidad de los resultados en relación con el número de muestras, se ha analizado la correlación entre el porcentaje de la tasa de acierto del algoritmo y el ratio de muestras de entrada utilizadas para su procesamiento.

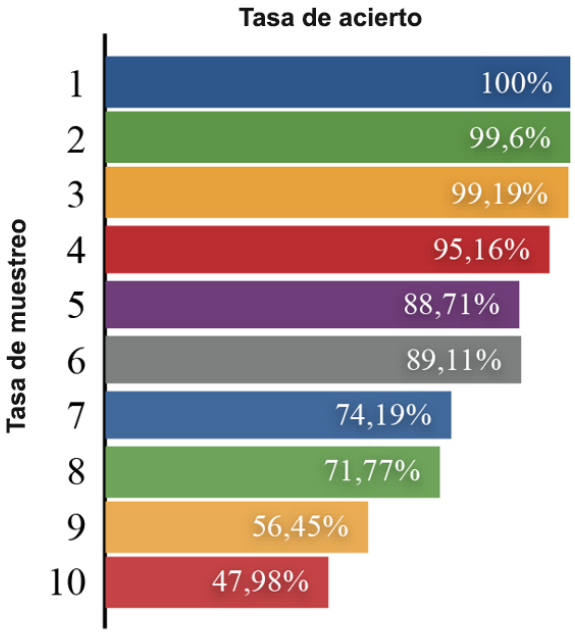


Figura 4.12: Relación entre la tasa de acierto y el ratio de muestreo, donde un ratio de muestreo de 1, quiere decir que para obtener el porcentaje de acierto mostrado se han utilizado todas las muestras y una tasa del 10, quiere decir que tan sólo se han utilizado la décima parte del número de muestras totales.

Tal y como se observa en la figura 4.12, la tasa de acierto se reduce considerablemente conforme disminuye el número de muestras utilizadas como entrada al algoritmo. No obstante, a la vista de estos resultados obtenidos con el hardware especificado, en casos de una limitación computacional muy alta, sería factible plantear una reducción del uso del número de muestras a la tercera o cuarta parte, ya que ello supondría una reducción de más de la mitad del tiempo de cómputo requerido y

tan sólo una reducción del porcentaje de la tasa de acierto de menos de un 5 % del total.

Con el fin de analizar en profundidad el coste de cada fase del algoritmo, la figura 4.13 muestra la distribución del coste para cada etapa de acuerdo con el número de muestras utilizadas. Por ejemplo, para 684 muestras (ratio 1), el tiempo total del algoritmo es de 0.058 segundos, donde la etapa de adquisición dura 0.00464 s. (un 8 % del tiempo total), el algoritmo de inferencia en líneas tarda 0.0377 s. (un 65 %), el filtrado, la detección de esquinas y la localización 0.00116 s. (2 %) y la función de coste ocupa el resto del tiempo, 0.0145 (25 %). Por otro lado, para 66 muestras (un ratio de 10), el tiempo total es de 0.044 s, donde 0.00136 s se consumen en el cálculo de la función de coste (31 %), 0.00015 s. en el filtrado, la detección de esquinas y la localización (3.5 %), 0.00242 s. en el algoritmo de inferencia de líneas (55 %) y 0.00046 s en la etapa de adquisición (10.5 %).

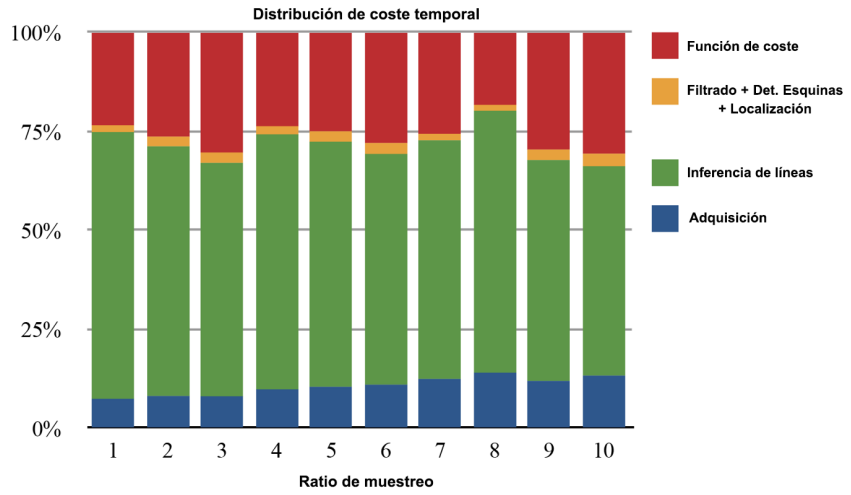


Figura 4.13: Distribución del coste de cada etapa del algoritmo $GEMA^2$ en función del ratio de muestras utilizado.

Analizando el gráfico completo, destaca la etapa de inferencia de líneas por ser la más costosa temporalmente con una media del 65 % del tiempo total. La fase de la función de coste ocupa una media del 25 % y la adquisición un 10 %. El filtrado, la extracción de esquinas y la localización son las fases más rápidas con un 2 % del tiempo total de ejecución del algoritmo.

Para representar estos resultados, ha sido necesario estimar la precisión de $GEMA^2$ y para ello se ha utilizado a priori un láser 3D (figura 4.14) para obtener medidas precisas y compararlas con los resultados obtenidos con el algoritmo.

Figura 4.14: (Izq)Escáner 3D para la obtención de las medidas del entorno utilizadas para medir la precisión del algoritmo $GEMA^2$.(Der) Nube de puntos 3D obtenida.

Sin embargo, este proceso resulta demasiado tedioso para realizarlo en cada escena real donde se aplica el algoritmo. Por este motivo se ha implementado de manera offline el conocido algoritmo de correspondencia ICP (Iterative Closest Point), el cual utiliza tantas iteraciones como sea necesario, hasta converger en la posición correcta del robot, utilizando las mismas medidas de entrada que $GEMA^2$. Al comprobar que todos los puntos emparejados en la última iteración del algoritmo ICP son correctos, se asegura que la solución alcanzada es la óptima global, por lo que es posible utilizar dichos datos para evaluar la precisión de los resultados de $GEMA^2$. La tabla 4.1 muestra dicha evaluación de la precisión en diferentes escenarios: entornos libres de objetos atípicos (más conocidos como *outliers*, los cuales se corresponden con objetos que no deberían estar ahí por no estar reflejados dentro del mapa conocido a priori), entornos con gente moviéndose en frente del robot, un escenario con *outliers* de tipo mobiliario y, finalmente, un escenario con *outliers* a modo de esquinas móviles (lo cual puede resultar crítico para $GEMA^2$).

Por otro lado, para evaluar la robustez del algoritmo, se han realizado varias pruebas empíricas en escenarios con distintos *outliers*, los cuales se pueden clasificar principalmente en dos grupos, tal y como muestra la figura 4.15:

- *Outliers de tipo 1.* Provocan rupturas de paredes rectas infiriendo en distintas rectas las cuales deberían formar una única línea.
- *Outliers de tipo 2.* Situados de forma perpendicular a los muros, pueden interpretarse como falsas paredes, las cuales pueden generar falsas esquinas originando una mala interpretación del entorno.

Un post-procesamiento de la extracción de líneas y la fusión de segmentos alineados podría atenuar los efectos de los *outliers de tipo 1.* Sin embargo, las pruebas empíricas han demostrado que el algoritmo $GEMA^2$ responde de una manera muy robusta frente a este tipo de *outliers* sobretodo cuando se utilizan todas las muestras recibidas del láser, tal y como refleja la tabla 4.1 de resultados. Las pruebas realizadas para evaluar este caso han incluido *outliers* estáticos y dinámicos mientras el robot permanecía quiero o en movimiento. Cabe destacar que

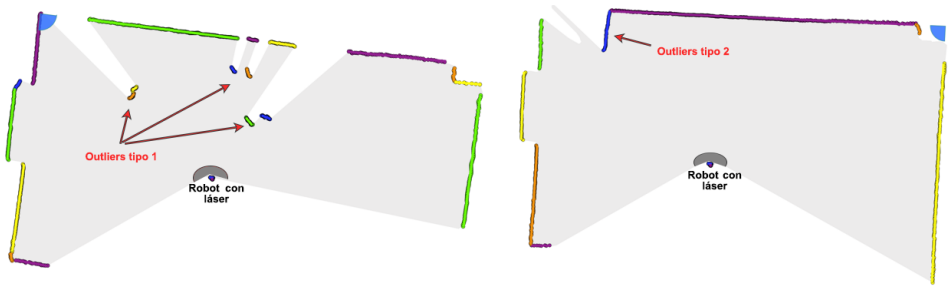


Figura 4.15: (Izq.) La presencia de 3 personas provoca que la pared del fondo se divida en 4 líneas rectas distintas. (Der) El *outlier* perpendicular de color azul (una caja apoyada al fondo de la pared) es malinterpretado como una pared que realmente no existe.

el algoritmo no tiene una realimentación de los resultados obtenidos en iteraciones anteriores por lo que no distingue entre *outliers* dinámicos o estáticos.

Para el caso de los *outliers de tipo 2*, el hecho de malinterpretar una pared que en realidad no existe depende directamente de distintos factores como la sección del entorno visible por el sensor, el tamaño y la distancia del *outlier* respecto al sensor o la ambigüedad introducida por la posición de dicho *outlier*. Por ejemplo, si está situado en el centro de un espacio vacío o si forma un ángulo con respecto a otra línea o segmento detectado que tampoco existe dentro del mapa conocido, el riesgo de malinterpretar ese *outlier* es realmente bajo ya que la técnica de filtrado, desechará fácilmente esas muestras.

La siguiente tabla muestra los resultados obtenidos en distintos escenarios de evaluación considerando para todas las pruebas el total de muestras recibidas (ratio de muestras = 1). La tasa de aciertos mide la robustez de $GEMA^2$ ya que un resultado tan sólo se considera correcto si se ha realizado correctamente la detección de las esquinas. La precisión se ha calculado como la distancia Euclídea entre la posición estimada y la posición correcta, teniendo en cuenta para ello, tan sólo los resultados correctos.

Tipo de escenario	Tasa de aciertos	Precisión (cm.)
Sin outliers	100 %	3.16
Gente paseando	97.67 %	7.87
Mobiliario	88.66 %	5.13
Outliers de tipo 2	95.99 %	5.75

Tabla 4.1: Resultados conseguidos a través de distintos escenarios con un ratio de muestras = 1.

Una muestra de los resultados experimentales reales en distintos escenarios donde intervienen *outliers* puede visualizarse en el siguiente vídeo o capturando el código QR de la imagen derecha: `/https://www.youtube.com/watch?v=_99WzN-YehI`.

Respecto a estos resultados hay que tener en cuenta, como ya se ha mencionado anteriormente, que $GEMA^2$ no posee ningún tipo de realimentación iterativa. Debido a esto, en cada iteración la posición anterior estimada es ignorada para calcular otra nueva. De este modo el algoritmo no acumula ningún tipo de error entre iteraciones y no necesita el conocimiento de ningún estado anterior. Considerar las estimaciones previas podría acelerar aún más la ejecución del algoritmo ya que se podría evaluar tan sólo la parte del mapa conocido que esté cercana a la última posición conocida del robot. Esta funcionalidad se considerará para trabajos futuros además de la extensión al reconocimiento de otras formas primitivas geométricas distintas a las líneas.

4.6 Conclusiones del capítulo

En este capítulo se ha expuesto un nuevo algoritmo de autolocalización global en interiores, basado en correspondencias geométricas para la mejora del posicionamiento de robots móviles con recursos limitados:

- Se ha presentado un nuevo algoritmo de autolocalización global geométrica denominado GEMA^2, orientado a la localización global de robots de recursos limitados en entornos interiores.
- Se ha expuesto la metodología detallada que compone el algoritmo de correspondencia con inferencia en líneas y sus distintas fases de ejecución.
- Se ha descrito la plataforma omnidireccional de recursos limitados Robotino, sobre la que se ha implementado GEMA^2 consiguiendo resultados satisfactorios.
- Se ha realizado un análisis sobre el tiempo de cómputo requerido por cada una de las fases del algoritmo obteniendo resultados permisivos para plataformas de recursos limitados.
- El algoritmo $GEMA^2$ es aplicable de manera individual a cada robot que cumpla con los requisitos especificados y ofrece la flexibilidad necesaria como para poder ejecutarse tanto en cada tiempo de ciclo del bucle de control, como de manera puntual en los momentos que se requiera.
- Mediante pruebas experimentales, se ha demostrado la robustez y fiabilidad del algoritmo frente a outliers o cambios dinámicos del entorno.

Capítulo 5

Autolocalización Local y Global por Eventos mediante Filtros de Fusión Sensorial

En el capítulo 3 se desarrollaron distintos algoritmos de fusión sensorial orientados a la autolocalización local de robots en interiores y exteriores mediante una estrategia de corrección continua. El siguiente paso para mejorar la localización consiste en añadir a la estimación una corrección global, como puede ser el algoritmo GEMA^2 descrito en el capítulo anterior.

Si en la autolocalización local, el posicionamiento se realiza sobre desplazamientos o giros relativos a posiciones anteriores conocidas, el posicionamiento global se realiza de una manera absoluta respecto a un origen de coordenadas fijo y externo. No obstante, dicha integración no es trivial, sobre todo cuando se trabaja con sistemas de recursos limitados donde en ocasiones no es posible actualizar en cada ciclo de control la posición global ya que el tiempo de obtención de los datos o las medidas supera el periodo de muestreo. Conseguir un buen control en este tipo de sistemas de computación limitada conlleva mantener un periodo de muestreo rápido por lo que el tiempo de acceso a la información de los sensores o el tiempo de procesamiento o interpretación de dicha información, entre otras causas, suelen ser los motivos principales que provocan dicha problemática.

En este capítulo se expone una estrategia de actualización de la autolocalización global por eventos basada en los filtros de fusión sensorial descritos en el capítulo 3. La estrategia por eventos no intenta procesar la información global de manera continua como ocurre con la autolocalización local, sino que se plantea una actuali-

zación puntual de la *postura* global, acotando de esta manera el error de posición global y al mismo tiempo, el error de estimación local.

5.1 Problemática relativa a la obtención de la medición del posicionamiento global

Los esquemas de fusión de corrección continua presentados en el capítulo 3 parten de la presunción de que las mediciones de todos los sensores están disponibles para cada ciclo del periodo de muestreo y que además su obtención no supone un retardo significativo que no permite cumplir con el tiempo de muestreo del bucle de control. Tanto el retraso de la medida como la superación del tiempo de muestreo son problemas críticos que pueden desembocar en una mala autolocalización.

Para el caso de interiores existen diversos métodos de obtención de esta localización global como por ejemplo el reconocimiento de patrones o códigos QR posicionados en coordenadas conocidas del entorno, el posicionamiento de la cámara cenital a través de la detección de un señuelo, métodos de aproximación por Montecarlo, los conocidos algoritmos de SLAM o incluso el algoritmo de localización global geométrica descrito y propuesto en la sección 4.1, GEMA^2. En el caso de escenarios exteriores lo más común suele ser utilizar el posicionamiento GPS de antenas receptoras de satélites.

Cuando se trabaja con sensores avanzados o agentes externos es difícil garantizar que ninguno de los dos problemas vaya a ocurrir si se lleva a cabo la estrategia de corrección continua en el algoritmo de fusión. Por ejemplo, en interiores, el uso de cámaras para la detección de señuelos de posicionamiento requiere un tiempo de procesamiento de imagen inherente, el cual va a repercutir en un retardo entre el tiempo en el que se tomó la imagen y el tiempo en el que se calculó finalmente la posición global del vehículo. Si mientras se procesaba la imagen, el vehículo estaba en movimiento, la posición global calculada se corresponde con un instante anterior al actual, lo cual perjudica seriamente la fiabilidad de la localización.

Para el caso de sistemas de recursos limitados como el Lego Mindstorms NXT, la metodología utilizada para obtener un posicionamiento global ha sido la comunicación con un agente externo que gestiona la cámara cenital que gobierna el escenario por donde se mueve el robot. La cámara cenital utilizada en este trabajo puede capturar a una velocidad de 33 ms por imagen, a lo cual hay que añadir un tiempo mínimo de procesamiento de 10-15 ms por imagen y además sumar el tiempo de vuelo de las comunicaciones entre el sistema NXT y el agente externo. En este caso se tratan de comunicaciones Bluetooth 2.0 por lo que la tasa media de transferencia es aproximadamente de unos 100ms por envío. Dado que el periodo de muestreo utilizado en el control del NXT es de 50ms, resulta imposible conseguir una frecuencia que permita llevar a cabo la corrección continua para este caso.

En los casos en los que se trabaja con sistemas con mayor capacidad de cómputo como el robot *Summit XL* o el *Rbcar*, se ha podido instalar un sensor de posicionamiento global GPS. Este tipo de sensores son capaces de funcionar a altas frecuencias, sin embargo en algunas situaciones pueden presentar errores de disponibilidad como cuando el número de satélites conectados es reducido o nulo. Lo cual puede ocurrir al pasar entre edificios altos o cuando la antena GPS no consigue una buena cobertura debido a interferencias de radio.

En definitiva, la frecuencia de estos sensores de localización global, por lo general suele ser menor a la de los sensores embarcados para el cálculo de la localización local y por ello no pueden ser incorporados en todo instante k del periodo de muestreo. En consecuencia, a la hora de realizar su integración en los algoritmos de fusión sensorial hay que tener en cuenta: la frecuencia a la que es posible leer las mediciones y el ancho de banda disponible en las comunicaciones si la fuente de información se corresponde con un agente externo.

Existen algunas soluciones en la literatura aplicadas a este tipo de problemas con retardos. Por ejemplo, para casos donde existe un retardo conocido dentro de las mediciones, hay trabajos que retroceden la estimación actual al instante en el que se tomó la medida global para realizar la fusión en ese instante y luego volver a propagar la estimación en el tiempo al realizar la fusión para todos los instantes hasta el actual [16, 166]. Sin embargo, este método requiere una gran cantidad de memoria para almacenar todos los estados y las mediciones intermedias entre el instante de medición y el actual, además de requerir la definición de ecuaciones y matrices de dimensiones variables. Para casos donde existen varias las frecuencias de medición distintas pero con retardos conocidos y casi constantes, es posible recurrir a métodos de fusión sensorial multifrecuencia como exponen los siguientes trabajos [59, 49, 8, 11, 138]. Dichos desarrollos tampoco resultan factibles a la hora de implementarlos en plataformas donde el ancho de banda de las comunicaciones no es constante y la disponibilidad de las mediciones tampoco, por lo que no resultan adecuados.

Con el fin de poder incorporar estas mediciones globales con retardos y a una frecuencia variable, se plantea una solución basada en la teoría de control y muestreo basado en eventos para modificar los filtros de fusión expuestos previamente. En general, estos métodos realizan el filtrado y la fusión sensorial únicamente cuando un evento definido se activa o supera cierto umbral por lo que permiten definir esquemas de control que incorporen mediciones con frecuencias variables o puntuales.

5.2 Metodología de Fusión Sensorial basada en Eventos para la Autolocalización Global de Robots

Vista la problemática de la obtención de la medición de la autolocalización global, se decide dividir la localización en dos niveles distintos. Un nivel debe mantener la localización local con corrección continua tal y como se describió en el capítulo 3, accediendo a los sensores locales, realizando la fusión sensorial y localizando correctamente la plataforma. El otro nivel debe incorporar al algoritmo la medición puntual de la localización global, solamente cuando un evento determinado se dispare. De este modo, las mediciones $z_{k,v}$ será utilizadas en cada instante k por el filtro local quien estimará la *pose* del robot utilizando únicamente esta información. Por otro lado, la corrección del filtro global se realizará si el evento cumple su condición o supera su umbral de activación, en cuyo caso se realizará una solicitud de zk, p al agente externo utilizando los recursos necesarios de comunicaciones. Mientras se recibe una respuesta (siempre existirá un retardo inherente a estas comunicaciones) se mantendrá en ejecución el filtro local hasta que se reciba la medición zk, p. Una vez recibida, se debe compensar el tiempo de retardo y ejecutar la parte de corrección del filtro global, donde al finalizar se volverá a inicializar el valor del evento, desactivando la corrección global hasta que se vuelva a disparar el evento. La figura 5.1 muestra un esquema del lazo de control modificado para incorporar la solución propuesta. La línea discontinua indica que el acceso a la información no se realiza para cada instante k, sino que depende de un evento que se ha definido como R_A.

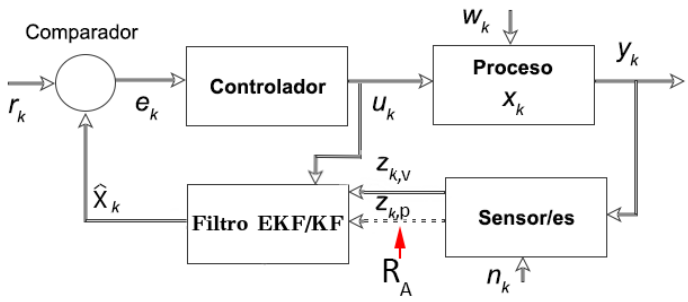


Figura 5.1: Lazo de control realimentado con el filtro de fusión basado en el evento R_A.

5.2.1 Estudio del Evento para la Corrección de la Localización Global

En las configuraciones de otros trabajos del mismo campo, se suele definir el evento R_A en base al error de estimación cometido al realizar el filtro de fusión, o bien se activa el evento cuando se da un cambio en el valor de la variable medida. Concretamente para casos de localización de robots, el error de estimación normalmente es desconocido para el robot a menos que constantemente se esté realizando consultas al sensor global. Lo cual no es deseable ya que puede provocar una saturación del canal de comunicaciones. Además, la postura del robot es una variable en constante evolución según la plataforma va desplazándose por el entorno. Definir un evento respecto a un cambio en el valor de la postura produciría una actualización para cada consulta que se realizase. Por estos motivos, conviene definir un evento adecuado que no consuma recursos de comunicaciones, que no dependa de ningún agente externo, ni necesite un tiempo excesivo de cómputo para plataformas limitadas.

Otros métodos de la literatura como [74] y [159] utilizan el error de navegación para definir el evento para el algoritmo de estimación por eventos (en las fases de predicción y corrección) aplicado a problemas de consenso en sistemas multi-agente. Se define un umbral de error de navegación, el cual en caso de ser superado, es cuando se realiza la corrección. No obstante, esta definición del evento tiene un gran inconveniente ya que el error de navegación (que indica si el robot ha llegado al punto de destino o a un punto intermedio de la trayectoria que debe seguir) puede ser independiente del error de localización (que indica con qué precisión se determinó la *pose* del robot o si el robot se ha perdido). Por ejemplo, puede ocurrir que un robot tenga un error de localización muy alto (el robot ha estimado una *pose* que difiere mucho a la real) pero que al mismo tiempo el error de navegación sea muy pequeño, es decir, la posición estimada erróneamente se encuentra casualmente cerca del punto objetivo que quería alcanzar. Por esta razón el error de navegación no puede formar parte del evento que lance la corrección global dentro del algoritmo. Además, un robot que por ejemplo se encontrase muy lejos de su objetivo (error de navegación muy alto), haría que el evento sobrepasase el umbral constantemente y se produjera una solicitud continua de las medidas de posición globales en cada instante k, provocando una saturación de las comunicaciones o de los procesos, hasta que el robot consiguiese acercarse lo suficiente.

Analizados estos trabajos previos, queda en evidencia la necesidad de redefinir el concepto de evento para el caso que aborda este capítulo.

5.2.2 Definición Conceptual del evento: Consciencia de Mal Posicionamiento

Conceptualmente se propone dotar de una *consciencia* individual de mal posicionamiento a cada uno de los robots o agentes que intervienen en el sistema. Es decir, individualmente cada robot debe realizar un control local de posicionamiento basado en filtros de fusión sensorial, pero además él mismo debe ser consciente del error de estimación local en la postura del robot $P_{k,p}$ que va cometiendo en cada iteración debido a la integración recursiva de la metodología. De este modo se propone definir un umbral $R_{A,lim}$ el cual, cuando el robot internamente detecta que se supera, es cuando debe solicitar a un agente externo las medidas de posicionamiento global para corregir su posición y disminuir al mismo tiempo su *consciencia* de mal posicionamiento. Se propone pues realizar la definición del evento en base a la *covarianza* del error de estimación local de la *pose* $P_{k,p}$, la cual debe determinarse en la etapa local del filtro de fusión para cada instante de muestreo k. Al ocurrir el evento, el agente solicitará una medida global y corregirá y mejorará su posición, por lo que el valor de la covarianza siempre disminuirá después de una actualización global y R_A volverá a quedar por debajo del umbral $R_{A,lim}$ hasta que vuelva a necesitar posicionarse de nuevo. Este método es plausible para sistemas de recursos limitados y garantiza que cada robot o agente únicamente hará uso de las mediciones lentas y/o retrasadas cuando él considere que las necesita por estar excesivamente mal posicionado.

5.2.3 Definición Matemática del evento: Elipsoides de Incertidumbre

Definido conceptualmente el evento cabe especificar matemáticamente su definición. Para ello se recurre a la interpretación geométrica de la matriz de covarianza gaussiana la cual es posible representar mediante el uso de elipsoides de incertidumbre.

Según [166], el exponente de la función de densidad de probabilidad (*pdf*) de una variable aleatoria gaussiana X de orden n con una autocovarianza C_X, es el único término que depende del valor actual de X, siendo el término que multiplica al exponente una constante de normalización. De modo que para definir un contorno donde la *pdf* sea constante, es suficiente estudiar el exponente descrito en la ecuación 5.1.

$$\left(X - \bar{X}\right)^T C_X^{-1} \left(X - \bar{X}\right) = 1 \tag{5.1}$$

Si en la ecuación anterior se iguala el exponente a 1, siendo C_X una matriz simétrica y positiva (para el caso de localización), se obtiene la definición matemática de un elipsoide general de dimensión n, centrado en $\bar{X}$. La dirección de los ejes del elipsoide viene dada por los vectores propios de C_X y la inversa de la magnitud de

los semiejes al cuadrado es igual a los valores propios de C_X. El elipsoide definido corresponde al lugar geométrico de la ecuación 5.2, es decir, a todos los puntos X que satisfacen la ecuación y que describen un contorno de densidad de probabilidad *pdf* constante, debido a que la *pdf* es constante si lo es su exponente, lo cual se cumple al igualarlo a 1, según se demuestra en [171],[170].

$$\mathrm{MD}(X, \bar{X}; C_X) = \sqrt{\left(X - \bar{X}\right)^T C_X^{-1} \left(X - \bar{X}\right)} \tag{5.2}$$

Para el caso general, se hace uso de la distancia de *Mahalanobis* tal y como se define en la ecuación 5.3. Se observa claramente como el elipsoide de la ecuación (5.2) corresponde con el lugar geométrico de los puntos X que se encuentran a una distancia Mahalanobis unitaria de $\bar{X}$.

$$\left(X - \bar{X}\right)^T C_X^{-1} \left(X - \bar{X}\right) = n^2 \tag{5.3}$$

Para el caso específico de localización resulta conveniente la definición de elipsoides con una distancia Mahalanobis mayor a la unidad, los cuales se utilizan para afrontar problemas como el SLAM ([171],[170]). Estas figuras se conocen como elipsoides n-sigma ($n\sigma$) los cuales se definen en función de n según se muestra en la figura 5.2 y en la ecuación (5.4), la cual describe el lugar geométrico de todos los puntos X a una distancia Mahalanobis n de la media $\bar{X}$.

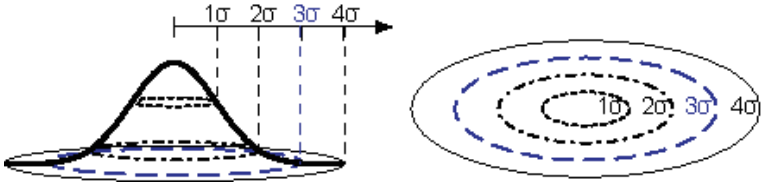


Figura 5.2: Descripción mediante los $n\sigma$ elipsoides de la matriz de covarianza para una variable aleatoria gaussiana bidimensional (2D) [171, 170].

$$\left(X - \bar{X}\right)^T C_X^{-1} \left(X - \bar{X}\right) = n^2 \tag{5.4}$$

Al integrar la *pdf* contenida dentro de los elipsoides se pueden establecer regiones de confianza. De esta forma, y tal y como se describe en [171, 170], pueden obtenerse las probabilidades porcentuales de que el valor actual (real) de la variable aleatoria X se encuentre dentro de su elipsoide $n\sigma$, obteniéndose para el caso 2D (2 dimensiones) una confianza de un $39,4\,\%$ para el elipsoide 1σ, un $86,5\,\%$ para el 2σ, un $98,9\,\%$ para el 3σ y un $99,97\,\%$ para el elipsoide 4σ. De estas regiones de confianza se observa que se requiere utilizar al menos los elipsoides 3σ

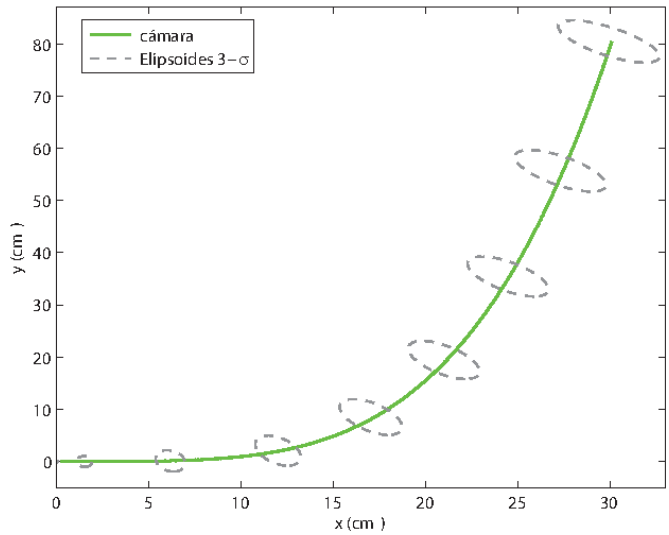


Figura 5.3: Evolución creciente de la covarianza del error de estimación indicada por medio de los elipsoides 3σ en un robot diferencial que estima su postura mediante odometría (encoders) y sigue una trayectoria circular; trayectoria medida utilizando una cámara cenital.

para representar adecuadamente la covarianza de la variable aleatoria X de forma gráfica.

A modo de ejemplo se muestra en la figura 5.3 la evolución de la covarianza de la estimación de la postura de un robot que utiliza únicamente la odometría a partir de los encoders para seguir una trayectoria circular. Se observa cómo según avanza el robot la incertidumbre representada por los elipsoides 3σ se incrementa debido a la acumulación del error. De esta forma la covarianza crecerá indefinidamente por lo que también lo harán los respectivos elipsoides 3σ hasta que se utilice una medición con información global.

Así pues queda justificada la conveniencia de utilizar la covarianza de la postura para definir el evento R_A, con la posibilidad de utilizar su representación geométrica dada por los elipsoides 3σ y aprovechar sus ventajas para poder seleccionar el límite $R_{A,lim}$ de forma intuitiva. Por ejemplo se puede describir el evento y su límite mediante características geométricas obtenidas a partir de los elipsoides 3σ. Para el caso de navegación 3D (robots voladores, robots submarinos, ...) se podría utilizar el volumen de los elipsoides, pero para el caso estudiado en la presente tesis y su aplicación a robots móviles, resulta más conveniente el uso del área para la definición de R_A, puesto que los robots se desplazan únicamente sobre el plano X,Y.

En esta tesitura se reescribe la ecuación 5.4 utilizando la nomenglatura del filtro y del error de estimación:

$$(x_{k,p} - \hat{x}_{k,p})^T P_{k,p}^{-1} (x_{k,p} - \hat{x}_{k,p}) = n^2 \tag{5.5}$$

Al utilizar el área definida de covarianzas X,Y no es necesario añadir la covarianza de θ ya que se incluye dentro del efecto del elipsoide, del mismo modo que para un modelo 3D el volumen del elipsoide incluiría las orientaciones correspondientes al modelo considerando únicamente la covarianza de posición en X,Y,Z.

Considerando la $P_{k,xy}$ del filtro local, se obtienen las magnitudes de los semiejes del elipsoide de la ecuación (5.5) como a_σ y b_σ al obtener los autovalores de $P_{k,xy}$, con $n = 3$ para el elipsoide 3σ, según se establece en la ecuación (5.6).

$$\begin{gathered} \mathbf{P}_{k,xy} = \begin{bmatrix} \sigma_x^2 & \sigma_{xy}^2 \\ \sigma_{xy}^2 & \sigma_y^2 \end{bmatrix}, T_\sigma = \sqrt{\sigma_x^4 + \sigma_y^4 - 2\sigma_x^2\sigma_y^2 + 4\sigma_{xy}^4} \\ a_\sigma = \sqrt{\frac{2l^2|P_{xy}|}{\sigma_x^2+\sigma_y^2+T_\sigma}}, b_\sigma = \sqrt{\frac{2l^2|P_{xy}|}{\sigma_x^2+\sigma_y^2-T_\sigma}} \end{gathered} \tag{5.6}$$

Al calcular el área de los elipsoides 3σ y dividiéndolos por el área del robot se obtiene un indicador normalizado del momento en el cual el error de la estimación de la *pose* del robot se relaciona en función de su tamaño en un plano. De esta forma, definiendo el evento $\boldsymbol{R}_A$ como el cociente de estas dos áreas es posible definir de manera intuitiva un límite $\boldsymbol{R}_{A,lim}$ adecuado para obtener una buena localización. La ecuación 5.7 refleja esta asignación.

$$\begin{aligned} A_{ellip} &= \pi a_\sigma b_\sigma \\ \boldsymbol{R}_A &= A_{ellip}/A_{robot} \end{aligned} \tag{5.7}$$

Así, el valor de R_A calculado para cada instante k está normalizado para cada robot según su tamaño y para todos significa lo mismo: qué área de incertidumbre se obtiene en función del área del robot. Por ejemplo, un R_A con valor 1.8 quiere decir que el área del error de posicionamiento se corresponde con 1.8 veces con el área del robot. Lo cual resulta muy intuitivo de imaginar.

Además, con la definición de la ecuación (5.7) se obtiene la ventaja de asociar el número de consultas al agente externo o sensor global, de acuerdo a la precisión conseguida por la estimación local, ya que si se tienen sensores muy precisos, se podría tardar un tiempo considerable en llegar al valor $R_{A,lim}$, con lo que se utilizaría con menor frecuencia el medio de comunicación y recursos según se defina el valor de $R_{A,lim}$. De esta forma, $R_{A,lim}$ se escoge como un compromiso entre el número de actualizaciones globales o solicitudes a agentes externos que se realizan

(consumo de ancho de banda, procesador, energía, etc) y la precisión deseada en la estimación de la postura del robot. Un valor alto de $R_{A,lim}$ requerirá una baja frecuencia de actualización global mientras que un valor bajo indicaría una fuerte restricción frente a incertidumbres y una frecuencia de actualización global más alta.

A continuación se muestra la modificación de los algoritmos EKF y KF de corrección continua 6 y 7 descritos en la sección anterior. Los cambios que se introducen se corresponden simplemente con una comprobación del valor del evento R_A para cada instante k. En caso de que dicho valor supere el límite del umbral establecido ($R_A > R_{A,lim}$), el algoritmo realiza la corrección global haciendo uso de $z_{k,p}$, reduciendo así el valor de $P_{k,p}$ y por tanto R_A, por lo que en la siguiente iteración no se cumplirá que $R_A > R_{A,lim}$ y únicamente el algoritmo trabajará con el filtro local hasta que el evento vuelva a dispararse. Los algoritmos 10 y 11 muestran el esquema del desarrollo de los mismos.

Queda a la vista de los algoritmos 10 y 11 que la modificación no supone una dificultad para su implementación en cualquier tipo de sistema que soporte la integración del filtro completo.

En la siguiente sección se expone la metodología desarrollada para la determinación del valor de $R_{A,lim}$.

5.2.4 Análisis para la Determinación del Valor del Evento

La especificación del valor exacto del evento se debe ajustar a cada caso según las necesidades de exactitud de la localización y las características de los robots o agentes que intervienen. No resulta igual de crítico en cuestión de consumo de ancho de banda, el acceso a los valores de un GPS embarcado en el mismo robot, que la solicitud de posicionamiento a una cámara externa gobernada por otro sistema.

Para estudiar este compromiso entre la mejora de la localización del esquema basado en eventos frente al uso del ancho de banda de las comunicaciones, es necesario realizar una serie de pruebas empíricas para recolección y análisis de datos. Para medir el buen desempeño del algoritmo se ha utilizado el índice **IAE**, el cual se define como la integral del valor absoluto del error de estimación del robot (en la posición x,y). Para medir el uso del ancho de banda de las comunicaciones se ha tenido en cuenta el número de solicitudes demandadas por el robot al agente externo o a la llamada de un posicionamiento externo, lo cual es regulado por $R_{A,lim}$. A modo de ejemplo demostrativo, a continuación se describe el estudio realizado para la plataforma LEGO Mindstorms NXT en configuración diferencial.

Se realizaron una serie de pruebas empíricas sobre el robot NXT de configuración diferencial. El vehículo realizó el seguimiento de una trayectoria cuadrada durante

Algoritmo 10: Algoritmo EKF recursivo en cascada con modelos local/global en cascada, asignación de entradas, predicción local de velocidades, modelo global de *pose* y corrección global basada en eventos

Entrada: $u_{k-1} = u_{Aks} = [\,u_1 \;\; u_2 \;\; u_3\,]^T$, $\hat{x}_{k-1,v}$,$P_{k-1,v}$, $\hat{x}_{k-1,p}$,$P_{k-1,p}$
Medición: $z_{k,v} = [\,v_{x,enc} \;\; v_{y,enc} \;\; \omega_{enc} \;\; \omega_{gyr} \;\; \omega_{comp}\,]_k^T$,$z_{k,p} = [\,x_k \;\; y_k \;\; \theta_k\,]_{GP}^T$
Datos: $f_v(\cdot)$ *y* $h_v(\cdot)$ *del modelo no lineal local* (B.50),$Q_{k,v}$,$R_{k,v}$,A_{robot},$R_{A,lim}$
$f_p(\cdot)$ *y* $h_p(\cdot)$ *del modelo no lineal global* (B.36),$Q_{k,p}$,$R_{k,p}$
Salida: $\hat{x}_{k,v} = \left[\begin{array}{cc} v_x & v_y \quad \omega \end{array}\right]_k^T$,$\hat{x}_{k,p} = \left[\begin{array}{ccc} x & y & \theta \end{array}\right]_k^T$,$P_{k,p}$, $P_{k,v}$
Inicialización: $\hat{x}_{0,v}$,$P_{0,v}$,$\hat{x}_{0,p}$,$P_{0,p}$
Para *el instante actual* k **hacer**
Predicción Local:
$\hat{x}_{k,v} = f_v\left(\hat{x}_{k-1,v}, u_{k-1}, 0\right)$,$A_{k-1,v} = \frac{\partial f_v}{\partial x_v}\Big|_{\substack{x_{k-1} \\ u_{k-1}}}$, $W_{k-1,v} = \frac{\partial f_v}{\partial w_v}\Big|_{\substack{x_{k-1} \\ u_{k-1}}}$
$P_{k,v} = A_{k-1,v} P_{k-1,v} A_{k-1,v}^T + W_{k-1,v} Q_{k-1,v} W_{k-1,v}^T$
$H_{k,v} = \frac{\partial h_v}{\partial x_v}\Big|_{\hat{x}_k,0}$, $N_{k,v} = \frac{\partial h_v}{\partial n_v}\Big|_{\hat{x}_k,0}$
Corrección Local:
$K_{k,v} = P_{k,v} H_{k,v}^T \Big(H_{k,v} P_{k,v} H_{k,v}^T + N_{k,v} R_{k,v} N_{k,v}^T\Big)^{-1}$
$\hat{x}_{k,v} = \hat{x}_{k,v} + K_{k,v}\left[z_{k,v} - h_{k,v}\left(\hat{x}_{k,v}, 0\right)\right]$
$P_{k,v} = \left(I - K_{k,v} H_{k,v}\right) P_{k,v}$
Modelo Global, Covarianza Global:
$\hat{x}_{k,p} = f_p\left(\hat{x}_{k-1,p}, \hat{x}_{k,v}\right)$, $A_{k,p} = \frac{\partial f_p}{\partial x_p}\Big|_{\substack{x_{k-1} \\ u_{k-1}}}$,$A_{k,pv} = \frac{\partial f_p}{\partial x_v}\Big|_{\substack{x_{k-1} \\ u_{k-1}}}$
$P_{k,p} = A_{k,p} P_{k-1,p} A_{k,p}^T + A_{k,pv} P_{k,v} A_{k,pv}^T + Q_{k,p}$
Elipsoide 3σ: utilizar $P_{k,xy}$ para calcular el evento R_A:
Obtener Semiejes del Elipsoide 3σ: a_σ,b_σ, ecuación (5.6)
$A_{ellip} = \pi a_\sigma b_\sigma$, $\Rightarrow \boldsymbol{R_A} = A_{ellip}/A_{robot}$, ecuación (5.7)
Si $R_A > R_{A,lim}$ **entonces**
Corrección Global: $H_{k,p} = \frac{\partial h_p}{\partial x_p}\Big|_{\hat{x}_k,0} = I_{3\times 3}$, ecuación (3.19)
$K_{k,p} = P_{k,p} H_{k,p}^T \Big(H_{k,p} P_{k,p} H_{k,p}^T + R_{k,p}\Big)^{-1}$
$\hat{x}_{k,p} = \hat{x}_{k,p} + K_{k,p}\left(z_{k,p} - H_{k,p}\hat{x}_{k,p}\right)$
$P_{k,p} = P_{k,p} - K_{k,p} H_{k,p} P_{k,p}$
fin
fin

Algoritmo 11: Algoritmo KF recursivo en cascada con modelos local/global en cascada, asignación de entradas, predicción local de velocidades, modelo global de *pose* y corrección global basada en eventos

Entrada: u_{k-1}=u_{Akns}=$[\,u_1 \;\; u_3\,]^T$=u_{dif}=$[\,a \;\; \alpha\,]^T$,$\hat{x}_{k-1,v}$,$P_{k-1,v}$,$\hat{x}_{k-1,p}$,$P_{k-1,p}$

Medición: $z_{k,v} = [\;\; v_{x,enc} \;\; v_{y,enc} \;\; \omega_{enc} \;\; \omega_{gyr} \;\; \omega_{comp} \;\;]^T_k$,

$z_{k,p} = [\;\; x_k \;\; y_k \;\; \theta_k \;\;]^T_{GP}$

Datos: A_k *y* B_k *modelo lineal local* (B.23) *(o* (B.52)*)*,$Q_{k,v}$,$R_{k,v}$,A_{robot},

$f_p(\cdot)$,$h_p(\cdot)$ *modelo no lineal global* (B.11),$Q_{k,p}$,$R_{k,p}$,$R_{A,lim}$

Salida: $\hat{x}_{k,v} = \left[\; v_x \quad v_y \quad \omega \;\right]^T_k$,$\hat{x}_{k,p} = \left[\; x \quad y \quad \theta \;\right]^T_k$,$P_{k,p}$, $P_{k,v}$

Inicialización: $\hat{x}_{0,v}$,$P_{0,v}$,$\hat{x}_{0,p}$,$P_{0,p}$

Para *el instante actual* k **hacer**

Predicción Local:

$\hat{x}_{k,v} = A_{k-1,v}\hat{x}_{k-1,v} + B_{k-1,v}u_{k-1}$

$P_{k,v} = A_{k-1,v}P_{k-1,v}A^T_{k-1,v} + W_{k-1,v}Q_{k-1,v}W^T_{k-1,v}$

Corrección Local:

$K_{k,v} = P_{k,v}H^T_{k,v}\left(H_{k,v}P_{k,v}H^T_{k,v} + N_{k,v}R_{k,v}N^T_{k,v}\right)^{-1}$

$\hat{x}_{k,v} = \hat{x}_{k,v} + K_{k,v}\left(z_{k,v} - H_{k,v}\hat{x}_{k,v}\right)$

$P_{k,v} = \left(I - K_{k,v}H_{k,v}\right)P_{k,v}$

Modelo Global, Covarianza Global:

$\hat{x}_{k,p} = f_p\left(\hat{x}_{k-1,p}, \hat{x}_{k,v}\right)$,$A_{k,p} = \left.\frac{\partial f_p}{\partial x_p}\right|_{\substack{x_{k-1}\\u_{k-1}}}$,$A_{k,pv} = \left.\frac{\partial f_p}{\partial x_v}\right|_{\substack{x_{k-1}\\u_{k-1}}}$

$P_{k,p} = A_{k,p}P_{k-1,p}A^T_{k,p} + A_{k,pv}P_{k,v}A^T_{k,pv} + Q_{k,p}$

$\approx P_{k-1,p} + A_{k,pv}P_{k,v}A^T_{k,pv} + Q_{k,p}$

Elipsoide 3σ: utilizar $P_{k,xy}$ para calcular el evento R_A:

Obtener Semiejes del Elipsoide 3σ: a_σ,b_σ, ecuación (5.6)

$A_{ellip} = \pi a_\sigma b_\sigma$, $\Rightarrow \boldsymbol{R}_A = A_{ellip}/A_{robot}$, ecuación (5.7)

Si $R_A > R_{A,lim}$ **entonces**

Corrección Global: $H_{k,p} = \left.\frac{\partial h_p}{\partial x_p}\right|_{\hat{x}_k,0} = I_{3\times3}$, ecuación (3.20)

$K_{k,p} = P_{k,p}H^T_{k,p}\left(H_{k,p}P_{k,p}H^T_{k,p} + R_{k,p}\right)^{-1}$

$\hat{x}_{k,p} = \hat{x}_{k,p} + K_{k,p}\left(z_{k,p} - H_{k,p}\hat{x}_{k,p}\right)$

$P_{k,p} = P_{k,p} - K_{k,p}H_{k,p}P_{k,p}$

fin

fin

3 minutos con valores distintos de $R_{A,lim}$, almacenando durante cada prueba la estimación de la *pose* realizada por el filtro KF del algoritmo 11, la medición de la localización global de una cámara cenital (L_GM) y el número de consultas realizadas a la misma a través de una comunicación Bluetooth con el agente externo que la gobernaba. Con esta información se obtuvo el IAE para cada prueba en posición x,y y el error promedio porcentual de ambas coordenadas (obtenido a partir del IAE de una ampliación de un 1 % en las trayectorias x e y). La figura 5.4 muestra la variación de estos datos respecto a $R_{A,lim}$.

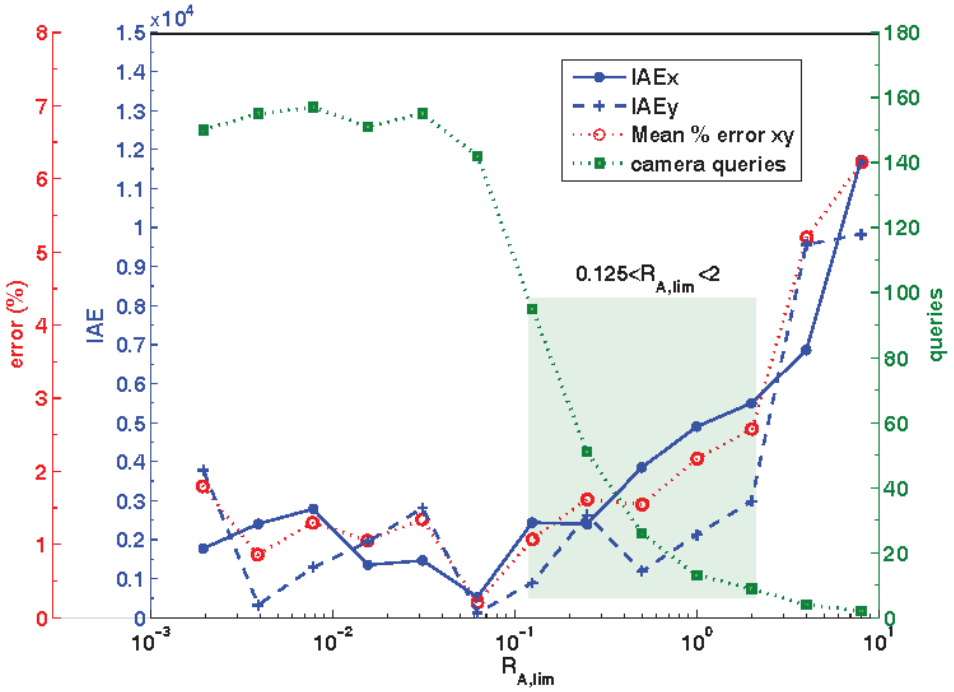


Figura 5.4: Relación entre $R_{A,lim}$, el índice IAE, el error promedio porcentual y el número de consultas al sensor global para el conjunto de pruebas de seguimiento de trayectoria en 3min, robot NXT diferencial.

Se observa claramente en la figura 5.4, que el índice IAE crece (mayor error de estimación de la *pose*) según aumenta el valor de $R_{A,lim}$, lo cual es predecible y razonable debido a que un límite con valor alto produce un menor número de actualizaciones globales por tardar más en alcanzar $R_{A,lim}$. Por otra parte, un valor bajo en $R_{A,lim}$ produce un mayor número de consultas a la cámara (más coste computacional y mayor retardo) y un incremento en la precisión del método, produciendo una disminución en el valor IAE hasta un valor límite que depende de la plataforma y de los recursos y retardos comunicación presentes. De hecho son estos factores los que limitan el número máximo de llamadas que se pueden

realizar, bien por saturación del ancho de banda o bien por un excesivo retardo en la obtención de los datos. En caso de los métodos tradicionales, se realizarían llamadas al sensor global en cada instante de muestreo, siendo posible obtener un menor IAE pero con la desventaja de no ser implementables en una plataforma con ancho de banda o recursos limitados.

De la figura 5.4 se ha sombreado en color verde el rectángulo de compromiso entre el uso del ancho de banda (consultas externas) y la precisión del método (índice IAE), el cual es controlado mediante la asignación del límite del evento $R_{A,lim}$. Se aprecia cómo se utilizan menos recursos y se realizan un menor número de llamadas si $R_{A,lim} > 2$, pero esto produce a su vez una menor precisión en la estimación con un IAE alto. Por el contrario, valores de $R_{A,lim} < 0.125$ producen un índice IAE reducido pero requieren de un mayor número de consultas al sensor global. De esta forma se puede concluir que el rango adecuado para el límite del evento puede establecerse en $0.125 < R_{A,lim} < 2$ para la plataforma LEGO NXT de configuración diferencial, de forma que se obtiene una solución de compromiso entre un valor IAE adecuado junto con un número de llamadas reducido.

Por último, cabe comentar que en la figura 5.4 también se muestra un eje adicional indicando el error promedio porcentual en las posiciones x e y, obtenidas en cada prueba utilizando como referencia el error IAE de una ampliación de un 1 % en las trayectorias x e y, para escalar los valores IAE del error de estimación. Se observa que para el rango $R_{A,lim}$ establecido como adecuado para el robot diferencial, se produce un incremento del error de un 1.1 % a un 2.6 % con una reducción de 95 a 9 consultas al sensor global en 3min. Si $R_{A,lim} > 2$, entonces el error puede incrementarse hasta un 6.2 % utilizando únicamente 2 consultas en 3min, y si $R_{A,lim} < 0.125$, entonces se obtiene un error mínimo para la plataforma diferencial de un 0.2 % pero utilizando una gran cantidad de recursos computacionales, realizando 142 consultas o más en los 3min de la prueba.

De este análisis se observa que el algoritmo con corrección basada en eventos KF (algoritmo 11), con un valor adecuado del $R_{A,lim}$, puede realizar una estimación de la postura con un valor aceptable del IAE (y en el error porcentual) a la vez que se ahorran recursos comunicaciones y ancho de banda, ya que únicamente se realizan las consultas necesarias al sensor global para corregir la estimación local. Este análisis puede realizarse al implementar el método en otras plataformas, con el fin de obtener el rango óptimo de $R_{A,lim}$ a utilizar según los recursos disponibles en el robot.

5.3 Resultados Obtenidos en Pruebas Empíricas

Se implementó el filtro descrito en el algoritmo 11 en distintas plataformas de recursos limitados analizando para cada caso el valor de compromiso del $R_{A,lim}$ adecuado para el buen desempeño del método. A continuación se describen los resultados obtenidos en las plataformas de interiores LEGO Mindstorms NXT en configuración diferencial y configuración Ackerman, así como para las plataformas de exteriores Summit XL y Rbcar.

5.3.1 Plataformas de Interiores

La arquitectura desarrollada para las plataformas interiores consiste en una red de sistemas multi-agente donde el robot LEGO Mindstorms realiza trayectorias dentro de un escenario gobernado por una cámara cenital capaz de localizar el robot gracias a un señuelo situado encima del mismo. La cámara está conectada a un computador (agente externo) el cual forma parte del sistema multi-agente a través de una comunicación vía Bluetooth. A continuación se exponen los resultados de las dos configuraciones utilizadas con el LEGO NXT, diferencial y ackerman.

5.3.1.1 Resultados en LEGO Mindstorms NXT configuración diferencial

Con la finalidad de poder comparar los resultados de la implementación del algoritmo 11 en LEGO NXT de configuración diferencial, se realizó la misma prueba de larga duración que se hizo para el algoritmo basado únicamente en odometría mediante encoders y el algoritmo 7 (KF) sin actualización global de posición.

La figura 5.5 muestra los resultados de la prueba donde en color rojo se ha representado la trayectoria monitorizada por la cámara y en color azul la primera trayectoria calculada internamente por el robot. Se observa una estimación adecuada y un desempeño notablemente mejor que en los algoritmos anteriores (véase figuras 3.11 y 3.12). Para esta prueba se escogió un valor de $R_{A,lim} = 0.5$ para cumplir con el compromiso calculado previamente y los resultados son muy favorables dado que la trayectoria no diverge en ningún momento y el error de estimación se mantiene acotado.

En la figura 5.6 se muestra la evolución de R_A durante los primeros segundos de ejecución. Como puede observarse, la covarianza del error de estimación representada por medio del área normalizada de los elipsoides 3σ, crece desde que se inicia la navegación hasta que se realiza la corrección con la información global, en cuyo momento se reinicia el valor de $P_{k,p}$ a su valor inicial $P_{0,p}$, y se disminuye R_A a su valor inicial.

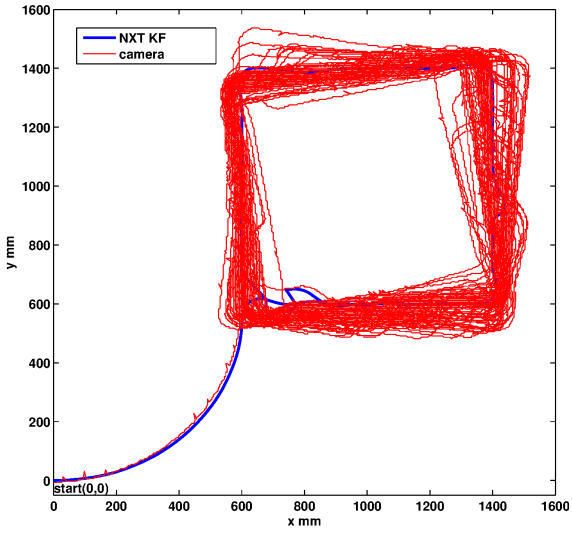


Figura 5.5: Prueba de larga duración, seguimiento de la trayectoria durante 30min, robot diferencial NXT con KF, Alg. 11 con L_{GM} y $R_{A,lim} = 0.5$

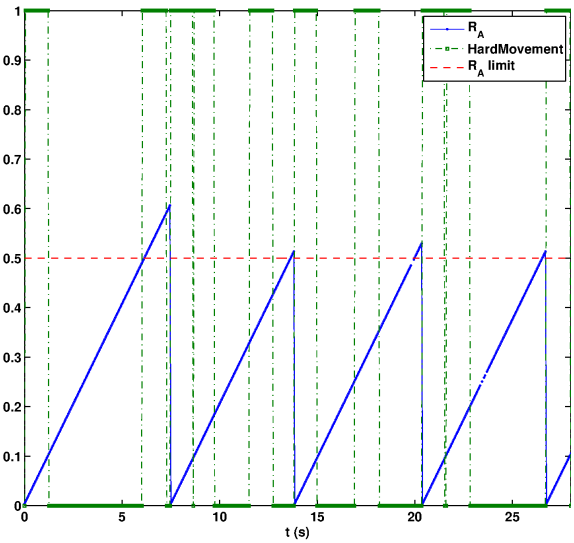


Figura 5.6: Prueba de larga duración, evolución del evento R_A, robot diferencial NXT con KF, Alg. 11 con L_{GM} y $R_{A,lim} = 0.5$

La información global es solicitada cuando $R_A > R_{A,lim} = 0.5$, por ejemplo en el tiempo 6.25s de la figura 5.6 (primera vez que se supera $R_{A,lim}$). En este instante, el robot comprueba la condición de movimiento brusco ($|e_{L,R}| > 2$), la cual resulta como "verdadera"(valor 1 en la figura 5.6), por lo que no se realiza la actualización global. Esta condición de movimiento brusco se añade debido a que las acciones de control resultantes de la actualización global pueden llegar a saturarse en algunos casos y perjudicar fuertemente el desempeño del filtro. Con esto, según los resultados, el algoritmo espera 1s y vuelve a comprobar la condición $|e_{L,R}| > 2$, que resulta ser "falsa"(valor 0 en la figura 5.6), por lo que el algoritmo procede a obtener la información global y a realizar la actualización (aproximadamente en el tiempo 7.45s) con lo que R_A vuelve a su valor inicial. De esta forma se logra acotar el error de estimación sin saturar el uso del ancho de banda de comunicación de la plataforma.

Para ver la comparativa de desempeño entre los tres algoritmos, se obtuvo el índice IAE y el promedio de los índices x e y, su equivalente en error porcentual y el porcentaje de mejora porcentual respecto a la estimación odométrica.

Tabla 5.1: Índice de desempeño IAE y errores porcentuales para la prueba de larga duración (30min), robot NXT diferencial

Pruebas en las figuras 3.11, 3.12 y 5.5	IAE			Error (%)	Mejora % respecto a la odometría
	x	y	Promedio xy	Promedio xy	Promedio xy
Odometría	$9.541e+05$	$6.750e+05$	$8.146e+05$	46.057	–
KF	$2.949e+05$	$4.376e+05$	$3.663e+05$	20.734	238.9
KF+Evento	$6.221e+04$	$5.342e+04$	$5.781e+04$	3.270	1398.8

En la tabla 3.1 se aprecia la mejora cuantitativa que ofrece el algoritmo de corrección global por eventos, el cual obtiene un índice de IAE cercano al 3 % frente al 46 % y 20 % que ofrecen el resto de algoritmos.

Se realizaron dos pruebas adicionales para verificar el buen desempeño del algoritmo. Una con una trayectoria espiral rectangular (5.7) y otra con una trayectoria cuadrada doble desplazada en diagonal (5.8). En las gráficas se ha representado la trayectoria monitorizada por la cámara, la trayectoria calculada internamente por el robot y los puntos donde se ha realizado una actualización global de posición.

Finalmente, en la dirección `/https://www.youtube.com/watch?v=b7oyDZrECTc` puede visualizarse un vídeo que resume las distintas trayectorias y la comparativa visual del desempeño entre los tres algoritmos.

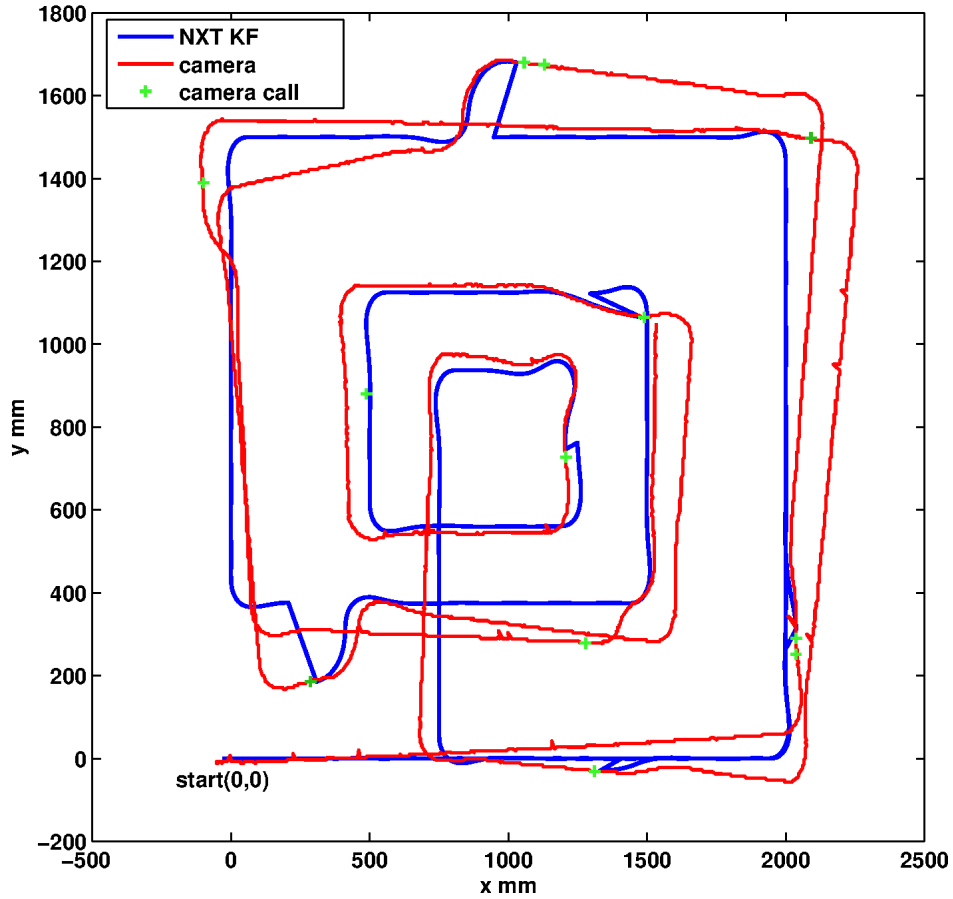


Figura 5.7: Prueba de desempeño, seguimiento de una trayectoria espiral rectangular durante 5min, robot diferencial con el filtro local, Alg. 11 con L_{GM} y $R_{A,lim} = 1.0$

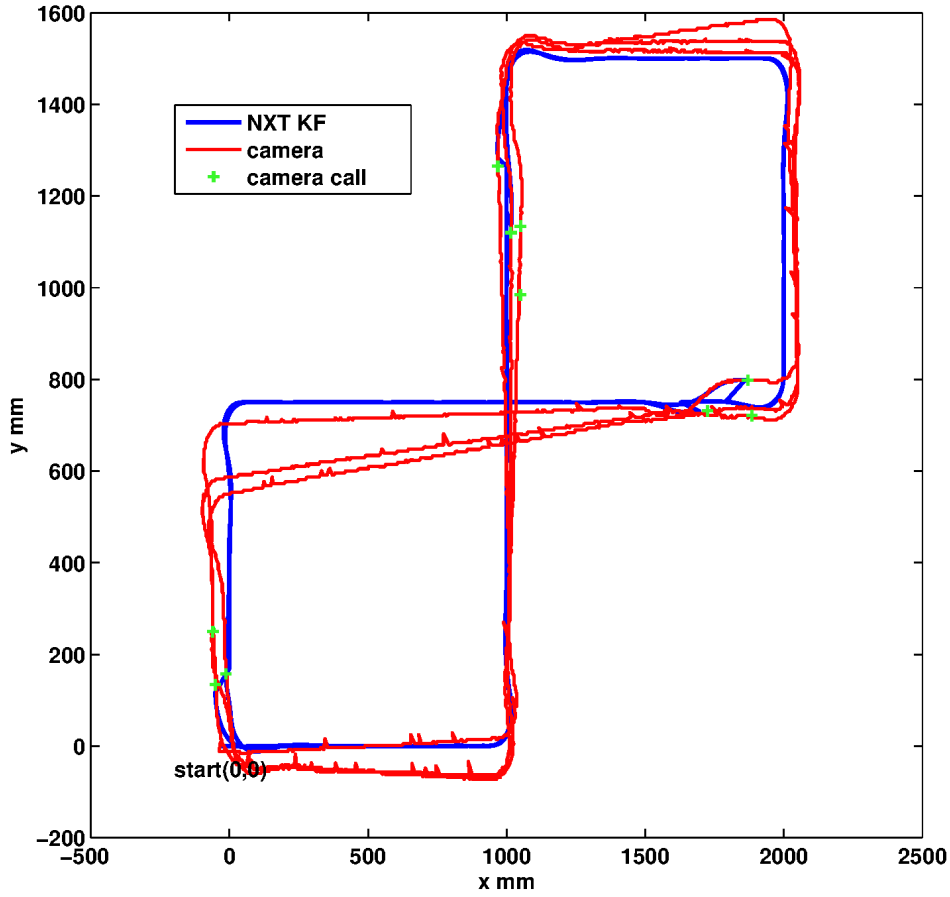


Figura 5.8: Prueba de desempeño, seguimiento de una trayectoria cuadrada doble en diagonal durante 5min, robot diferencial con el filtro local, Alg. 11 con L_{GM} y $R_{A,lim} = 1.0$

5.3.1.2 Resultados en LEGO Mindstorms NXT configuración Ackerman

Siguiendo el mismo procedimiento realizado con el robot diferencial, una vez implementado el algoritmo 11, se realizó una prueba de larga duración (30 min) con el robot en configuración Ackerman siguiendo una trayectoria rectangular.

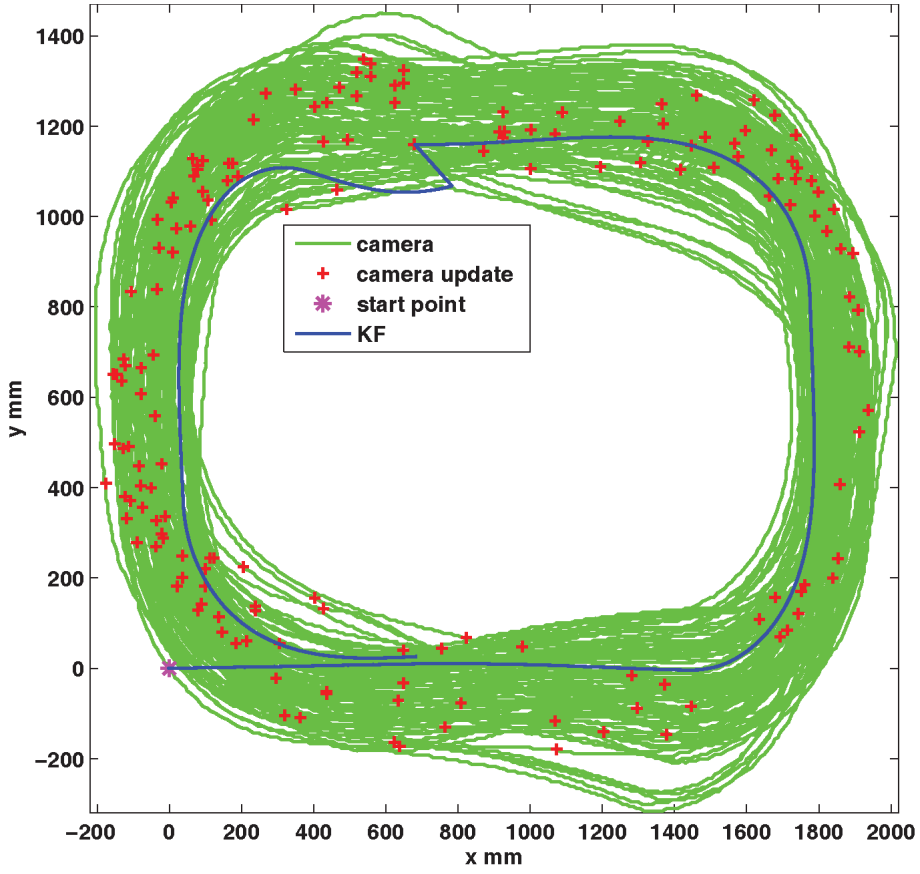


Figura 5.9: Prueba de larga duración, seguimiento de la trayectoria rectangular durante 30min, robot Ackerman navegando en interiores.

Se observa en la figura 5.9 la trayectoria capturada por la cámara cenital en color verde y la trayectoria calculada para la primera vuelta por el robot NXT. Además, se muestra en rojo los puntos donde el robot ha solicitado a la cámara su localización global.

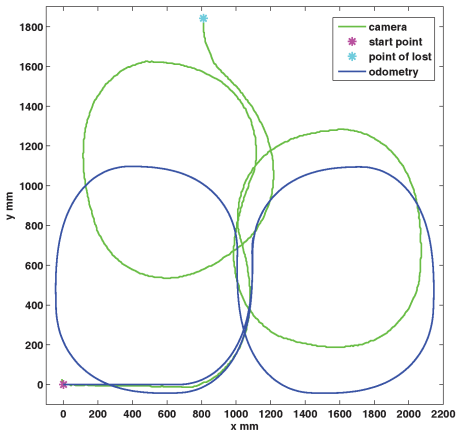
El valor de $R_{A,lim}$ para este caso ha sido de 1.0 y como puede observarse el algoritmo responde correctamente ya que el robot mantiene su trayectoria fiel a la referencia y su error no diverge como ocurría en las figuras 3.11 y 3.12.

Dado el buen resultado de la prueba anterior se realizó otra prueba con una trayectoria más compleja, la figura de un ocho cuadrado. En la figura 5.10 se recogen los resultados de los tres algoritmos a comparar: la odometría únicamente mediante encoders (figura 5.10(a)), la estimación del filtro KF sin posicionamiento global (figura 5.10(b)) y la estimación del KF con corrección global por eventos (figura 5.10(c)).

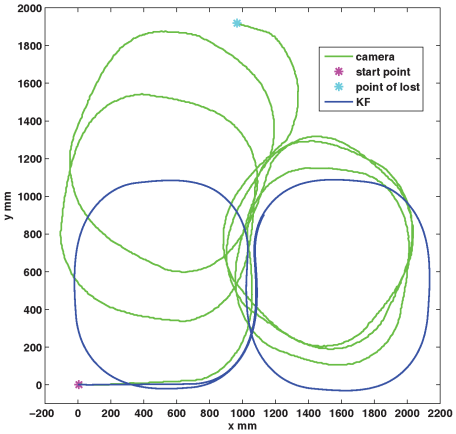
Como puede observarse en la figura, haciendo uso de la localización por odometría la prueba a penas alcanzó el medio minuto de duración, ya que el error de posicionamiento hizo que el robot saliera rápidamente del escenario capturado por la cámara cenital. En el caso del filtro KF sin corrección global, el robot consiguió mantenerse hasta un minuto dentro de la zona gobernada por la cámara, luego la localización mejoró significativamente. Sin embargo, haciendo uso del algoritmo de corrección global por eventos se observa cómo el robot es capaz de mantener la trayectoria sin desviarse ni salirse del escenario durante los 30 minutos que duró la prueba.

Visto el buen desempeño del filtro, se concluye la notable mejora en el posicionamiento del robot utilizando el filtro de corrección global basado en eventos. El robot solicita la medida global a la cámara cuando R_A basado en los elipsoides 3σ supera el límite del evento establecido en 1.2 para esta prueba. Bajo este esquema, la precisión de la estimación de la *pose* que ofrece el algoritmo es adecuada. El error no crece indefinidamente, sino que está acotado durante toda la ejecución de la prueba.

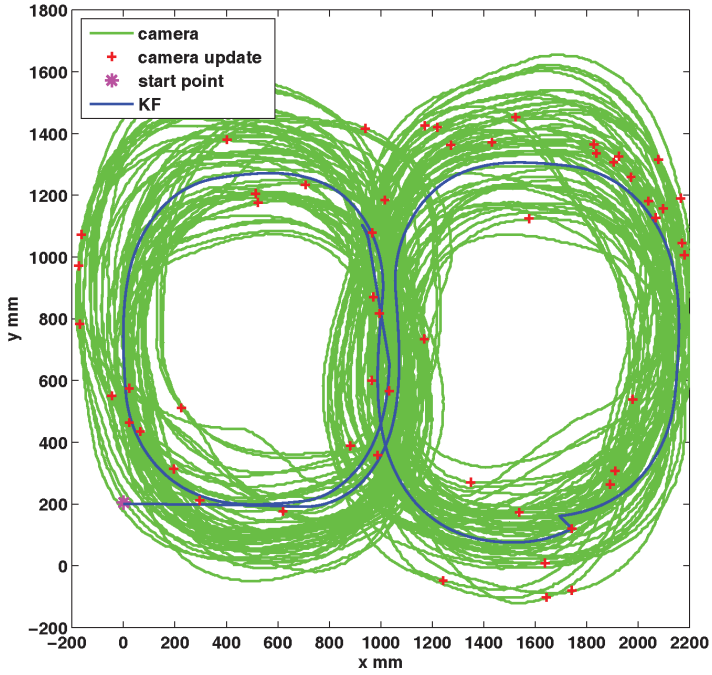
En la dirección `/https://www.youtube.com/watch?v=ZQ5b6OYLlqU` puede observarse un vídeo que resume todas las pruebas realizadas sobre el robot LEGO NXT en configuración Ackerman.



(a) Odometría



(b) KF



(c) KF+Evento, $R_{A,lim} = 1.2$

Figura 5.10: Prueba de larga duración, seguimiento de la trayectoria cuadrada doble durante 30min, robot NXT Ackerman.

5.3.2 Plataformas Exteriores

A diferencia de las plataformas interiores donde para la obtención de la posición global se ha hecho uso de una cámara cenital, en las pruebas exteriores la arquitectura es muy distinta desde el punto de vista hardware. En este caso, se utilizó un sensor GPS embarcado en el propio robot por lo que las comunicaciones entre el agente de control y el agente de posicionamiento global, no suponían un problema ni originaban ningún retraso. Aun así el esquema del filtro se implementó siguiendo el mismo patrón que con el LEGO NXT. Los robots mantenían su localización local de corrección continua y cuando el evento sobrepasaba el umbral $R_{A,lim}$, se leía el último dato recibido por el GPS.

Hay que recordar que en este tipo de plataformas la arquitectura distribuida hace que los procesos estén desacoplados, de modo que el control de localización se efectúa mediante un proceso independiente al proceso de medición de coordenadas GPS. También cabe recapitular que la comunicación entre procesos o agentes, aún siendo dentro de un mismo host, se realiza mediante tópics basados en sockets. Los topics es el medio por el que los procesos intercambian datos.

El problema principal para este caso, es que en el instante en el que se solicita la medida o la lectura del dato, puede haber ocurrido algún problema en la cobertura del GPS, o en la recepción de los satélites o que el error de la posición leída sea excesivamente alto debido a alguna interferencia.

Por este motivo es necesario que al recibir la medida se realicen una serie de verificaciones antes de aplicarla directamente a la fase de corrección. En concreto, se aplican dos cambios importantes respecto al LEGO NXT:

- El número de satélites en línea N_{sat} cuando se recibe la medida GPS debe superar un mínimo establecido ($N_{sat,min}$).
- La matriz $R_{k,p}$ (valor nominal de la varianza del sensor) del filtro se calcula dinámicamente, siendo inversamente proporcional al número de satélites en línea cuando se recibe la medida.

Estas condiciones pueden generar un mayor número de solicitudes de posicionamiento global, sin embargo en este caso las solicitudes no penalizan ni consumen ancho de banda puesto que el nodo de control sólo lee en el momento se publica el dato.

5.3.2.1 Resultados en Summit XL

En el robot Summit XL se instaló el GPS IG-500N utilizado en aplicaciones de vehículos aéreos no tripulados. Para poder monitorizar de algún modo la prueba, se instaló en el robot la antera receptora de un GPS diferencial RTK modelo 3i de la marca $VBOX$. Este GPS no se utilizó para la localización del vehículo sino que simplemente capturaba la posición para poder compararla con la del GPS IG-500N.

Para la prueba se realizó el recorrido en sentido horario de una pista deportiva de superficie plana utilizando el GPS IG-500N como fuente de información global. Al evento R_A basado en el área 3σ se le agregó la condición de que el número de satélites fuese de mínimo 6 ($N_{sat,lim} = 6$), lo cual en caso de no cumplirse volvería a solicitar una nueva medida.

Los resultados de la ejecución de la prueba se muestran en la figura 5.11.

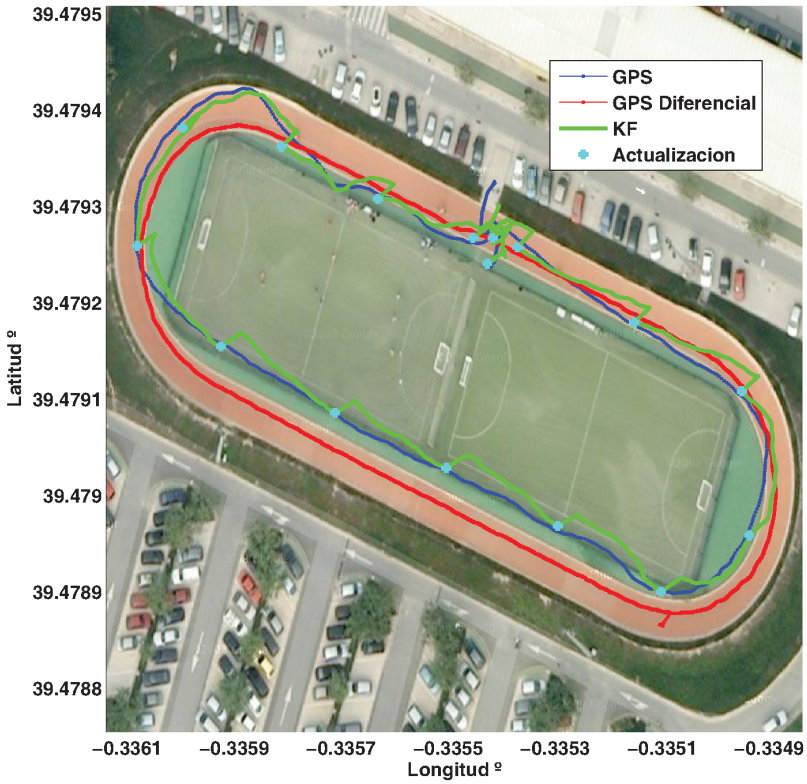


Figura 5.11: Prueba Summit XL en exteriores con el GPS IG-500N. Trayectorias georeferenciadas sobre la imagen satelital de la pista deportiva obtenida de Google Earth®.

La línea verde muestra la localización realizada por el KF con corrección global por eventos, la línea azul muestra la monitorización del GPS IG-500N y los puntos azules claros muestran los instantes en los que se produjo una actualización global. El GPS diferencial utilizado para monitorizar la prueba (línea roja) es mucho más preciso que el GPS IG-500N y muestra de manera más fiel la trayectoria de referencia de la plataforma.

A la vista de los resultados se observa un desempeño adecuado del filtro de fusión ya que la postura estimada por el filtro con corrección global por eventos, es similar a la medida del GPS diferencial. Conforme aumenta el desvío del error de estimación llega un punto en el que se genera el evento ($R_A > R_{A,lim} \& N_{sat} > N_{sat,min}$) y se produce la actualización utilizando la información global del GPS eliminando el error acumulado hasta el momento. Para observar la evolución del error de estimación la gráfica 5.12 muestra su evolución en valor absoluto durante la ejecución de la prueba.

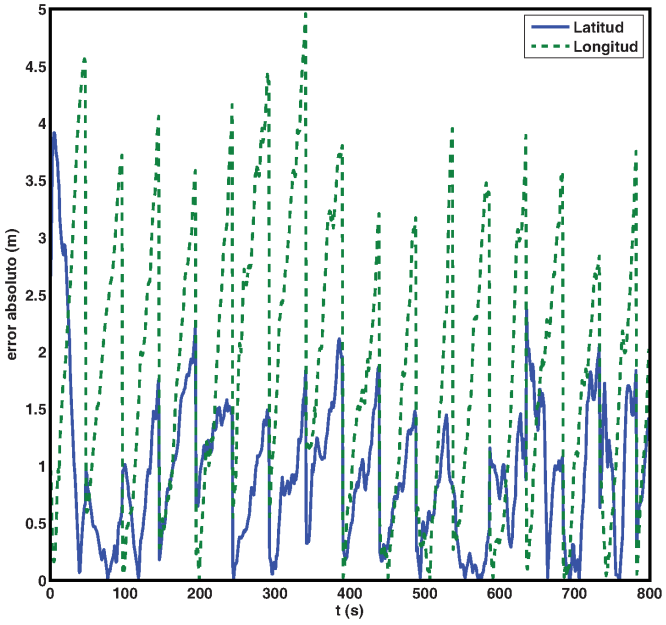


Figura 5.12: Prueba Summit XL en exteriores, evolución del error absoluto de estimación.

Se obtiene una precisión aceptable según la gráfica del error absoluto. Se identifican a simple vista los momentos donde se producen las actualizaciones globales y prácticamente el valor del error de estimación desciende a 0. Además el error está acotado y en ningún momento sobrepasa los 5 metros de incertidumbre. Este valor aún se podría hacer descender más utilizando un valor de $R_{A,lim}$ menor.

5.3.2.2 Resultados en Rbcar

En la plataforma Rbcar, en lugar de instalar el GPS IG-500N se decidió instalar directamente el GPS diferencial VBOX ya que la plataforma ofrece unas dimensiones plausibles para ello. Esto desembocó en una mejora considerable de la estimación de posición ya que el error inherente del GPS diferencial es mucho menor que el del GPS IG-500N utilizado con el Summit XL. La desventaja fue que al no contar con otra medida global de posicionamiento, no se pudo monitorizar el desempeño de la prueba para poder comparar los resultados obtenidos. Sin embargo, se proyectaron las trayectorias realizadas sobre el mapa de Google Earth® para verificar visualmente el desempeño del filtro.

La implementación del algoritmo fue exactamente la misma que para el Summit XL puesto que los dos incorporan la implementación del middleware JADE-ROS para sistemas distribuidos. Lo único que hubo que modificar fue el valor de $R_{A,lim}$ y $N_{sat,min}$ los cuales se establecieron en 1.5 y 7 respectivamente.

A continuación se muestran la pruebas realizada con el Rbcar por el recinto de un parking la Universidad Politécnica de Valencia.

Figura 5.13: Prueba Rbcar en exteriores con el GPS VBOX. Trayectorias georeferenciadas sobre la imagen satelital de un parking de la UPV obtenida por Google Earth®.

Se observa en la figura 5.13 cómo el vehículo se ciñe fielmente a la trayectoria realizada. El límite del evento establecido en 1.5 hizo saltar el evento a una frecuencia aproximada de 1.0 metros de recorrido, de ahí el buen desempeño. Además las condiciones meteorológicas y la zona escogida para la prueba fueron previamente analizadas para verificar que la cobertura GPS era la adecuada.

Demostrada la aportación de la presente tesis a la integración de la corrección global por eventos para la localización en sistemas distribuidos y de recursos limitados, se propone dar una vuelta más de tuerca a la problemática de la localización. Hasta el momento la fuente del posicionamiento global ha sido un sensor (cámara cenital, sensor y antena GPS, ...) especializado en esa labor. Sin embargo, en la siguiente sección se introduce la posibilidad de que el posicionamiento global también pueda ser recibido por cualquier robot o agente que sea capaz de posicionar a otro igual, es decir, que un mismo agente o robot sea la fuente que ofrezca la medida de posicionamiento global de otro agente. Bajo esta filosofía de localización de igual a igual se desarrolla el algoritmo de localización cooperativa que se describe en el siguiente capítulo.

5.4 Conclusiones del capítulo

En este capítulo se han expuesto las aportaciones y contribuciones de la presente tesis a la mejora de la autolocalización local y global de robots móviles con recursos limitados mediante el uso de sistemas multi-agente:

- Se ha analizado la necesidad de incluir una estimación de la autolocalización global de los robots para acotar y mejorar su *postura*.
- Se ha justificado la integración de la corrección de la autolocalización global mediante eventos, dadas las capacidades reducidas de las plataformas.
- Se ha realizado una definición del evento adecuada al caso de autolocalización de robots móviles, utilizando la covarianza del error de estimación como el conocimiento o la consciencia que cada robot tiene de sí mismo sobre su mal posicionamiento.
- A partir de la consciencia creada en cada agente sobre su mal posicionamiento, se ha implementado el algoritmo de fusión sensorial por eventos para corregir de manera puntual la postura del robot de manera global, disminuyendo así su error de estimación considerablemente y haciendo un uso mínimo del recurso de obtención del posicionamiento global.
- Se ha realizado un análisis para la determinación del valor del evento teniendo en cuenta el número de solicitudes al sensor global frente al error promedio porcentual, consiguiendo una solución de compromiso entre estos dos factores.
- Finalmente, se han presentado pruebas experimentales de la implementación de los métodos de localización sobre distintos tipos de robots tanto en entornos interiores como en entornos exteriores, con resultados satisfactorios que demuestran y validan las aportaciones de los algoritmos desarrollados en el presente capítulo.

Capítulo 6

Localización Cooperativa entre Robots mediante Sistemas de Control Distribuido

En un grupo de sistemas multi-agente de robots compartiendo un mismo escenario en el cual cumplen distintas misiones que tienen asignadas, es frecuente y necesario que los robots a menudo se detecten entre ellos para evitar colisionar entre sí. Bajo esta premisa y siguiendo la línea del algoritmo de localización global por eventos del capítulo anterior, surge la idea de desarrollar un algoritmo de localización cooperativa donde sean los propios robots quienes al detectarse entre sí puedan posicionar al resto de robots. La estrategia es totalmente complementaria e independiente al uso de los métodos anteriormente descritos y además responde a una filosofía típica de sistemas distribuidos donde cada agente trabaja de manera individual pero también se le asignan comportamientos colaborativos que mejoran una característica del colectivo: su localización en este caso.

Para ejecutar el algoritmo, los robots necesariamente deben estar equipados con un sensor que les permita detectar a los robots cercanos a él. Este sensor puede tratarse de un sensor de barrido láser, de infrarrojos, ultrasonidos o incluso una cámara. Todos estos sensores obtienen una medida *relativa* ya que devuelven la información respecto a la posición y orientación del robot que los incorpora.

Utilizando la postura y la covarianza de cada robot cercano detectado, junto con la medición relativa medida hasta cada uno, un robot puede estimar su *pose* mediante métodos geométricos, al considerar cada vecino como una baliza móvil de la cual conoce su posición y su covarianza (utilizando métodos de comunicación entre agentes) y su posición y orientación relativa respecto a él mismo (medidos por el

sensor). El resultado de esa localización se incorpora al filtro de fusión sensorial para mejorar la precisión de la estimación de la *pose*, lo cual asegura que mejorará la posición local estimada de cada uno de forma independiente según [151].

A parte del incremento de la precisión individual del posicionamiento, el esquema cooperativo utiliza una solución distribuida para realizar la localización en grupo sin requerir una centralización de ningún tipo. Además, en un escenario de robots heterogéneos donde exista un grupo de mejores prestaciones con acceso a información global y otro sin acceso a ella, este esquema permite distribuir y propagar esa mejora del acceso a la localización global a los robots que no tienen acceso a ella, incrementando así la precisión de su localización a pesar de todo.

Existen multitud de aplicaciones actuales de grupos de robots donde se requiere una gran precisión en la localización y la solución que se plantea en este trabajo puede ofrecer numerosas ventajas. Por ejemplo, en coordinación de equipos submarinos [15, 56], en grupos de robots aéreos y/o terrestres ejecutando misiones cooperativas [117, 48, 47, 192, 4] o en exploración de entornos desconocidos [145, 70, 146, 149].

El método propuesto de *Localización Cooperativa* requiere también un canal de comunicaciones entre todos los agentes que intervienen para ser efectivo. Tal y como se explicó en el estado del arte, la mayor parte de los métodos de localización multirobot que se dan en la literatura requieren la transmisión de gran cantidad de información utilizando de forma intensiva el ancho de banda de la plataforma. Por este motivo resultan métodos inadecuados cuando el acceso al medio de comunicación conlleva cierto retraso o es limitado (lo cual ocurre a menudo). Algunos métodos consideran algún tipo de limitación en el uso del ancho de banda como en [150, 125, 181, 102, 139] al reducir en alguna medida la cantidad de información transmitida o la frecuencia de intercambio de mensajes. Sin embargo, estos métodos requieren que la comunicación se mantenga estable entre todos los miembros del grupo por lo que no contemplan la posibilidad de que nuevos miembros se unan al grupo de manera dinámica, ni que se suprima algún integrante del equipo temporal o permanentemente. Además, estos métodos en general requieren gran cantidad de memoria y recursos computacionales, lo que dificulta su implementación en sistemas de recursos limitados.

Por estos motivos el algoritmo de localización cooperativa debe permitir mejorar la localización de cada miembro del sistema utilizando de forma eficiente los recursos computacionales y de comunicación de la plataforma. Para ello, se va a hacer uso de un método de actualización por eventos para incorporar la información relativa en el algoritmo de localización de fusión sensorial. De este modo se limita el uso de las comunicaciones ya que únicamente se transmitirá la información requerida por el método cooperativo cuando salte un evento que así lo determine.

Teniendo en cuenta los planteamientos anteriores, se expone a continuación el desarrollo del método de localización cooperativa para grupos de robots heterogé-

neos. En primer lugar se describe el marco de trabajo y las definiciones pertinentes del grupo de robots con localización cooperativa y su red de comunicación, así como el modelo que permite integrar las mediciones relativas en el esquema de fusión sensorial. Por último se expone el algoritmo cooperativo desarrollado con actualización cooperativa, en su versión basada en el tiempo o continua, y en su versión basada en eventos.

6.1 Descripción del Marco de Trabajo de la Localización Cooperativa

Para poder enfocar la problemática exclusivamente a la localización, se define un contexto detallado donde se define el concepto de *holarquía de robots cooperativos*, la especificación de la red de comunicación utilizada y el modelo de medición relativa utilizado.

6.1.1 Holarquía de Robots Cooperativos

La holarquía de robots cooperativos considerada en la presente tesis cumple con las siguientes condiciones:

- Se define un grupo de holones o robots G_R como el conjunto de N integrantes o miembros denotados como R_i, donde $i = 1, \ldots, N$ y $N \geq 2$. El número N es desconocido a priori $\forall R_i$, lo cual crea la necesidad de realizar un ajuste dinámico en las ecuaciones de fusión dependiendo del número de robots vecinos detectados por cada R_i en cada instante k.

- $\forall R_i \in G_R$ se define la *pose* como su posición (x, y) y su orientación θ respecto a los ejes de referencia global (X_G,Y_G), los cuales son comunes a todos los miembros en G_R. También cada R_i trabaja localmente mediante una referencia local (X_L,Y_L) que enmarca la medición de las velocidades locales (v,ω). En este marco, no se han considerado los casos de localización relativa local, es decir, en cualquier caso la localización relativa se realiza siempre en referencia a un eje global único para $\forall R_i$.

- Se considera G_R como un grupo de heterogéneo de robots, dividido en dos subconjuntos: un grupo de robots avanzados definido como G_{RA}, y otro de robots con recursos limitados, G_{RL}. Esto se debe a que cada $R_i \in G_R$ puede tener distintos sensores embarcados, distinto rango de detección D_r (distancia máxima a la que se puede detectar un vecino), distinto rango de comunicación C_r (distancia máxima en la que se puede mantener la comunicación con un robot vecino) y capacidad de cómputo y comunicación (recursos computacionales y ancho de banda).

- Cada $R_i \in G_R$ obtiene sus mediciones locales incluyendo las velocidades lineal v y angular ω desde los encoders $(v, \omega)_{enc}$, ω_{gyr} de un giróscopo, ω_{comp} de una brújula, además de las aceleraciones lineal a y angular α de los acelerómetros disponibles en la plataforma (u_1, u_2, u_3 en el caso con deslizamiento).

- Los robots pertenecientes a G_{RA} tienen acceso a algún sensor global (cámara cenital o embarcada, GPS, etc.), desde donde obtienen las mediciones globales $L_{GP} = (x, y, \theta)_{GP}$.

- Para la convergencia global del grupo se define que existirá al menos un miembro R_i con acceso a L_{GP}, según se justifica en [150]. Esta cantidad puede ser superior dependiendo del tamaño del grupo (N).

- Se definen las mediciones relativas como M_r, y se realizan en cada $R_i \in G_R$ por un sensor que obtiene la distancia ρ_{ij}, el ángulo relativo (orientación relativa) φ_{ij} y la identidad M_j de cada robot vecino R_j dentro del rango de detección D_r (constante) del robot R_i. Se considera que la medición relativa se puede realizar en toda la circunferencia del robot, con lo que el rango de detección del sensor relativo forma un círculo de radio D_r con centro en la posición del sensor sobre la plataforma móvil (por ejemplo en el centro geométrico o de masa del robot).

- El proceso de identificación de vecinos o *Matching* es el proceso mediante el cual se asigna la identidad del vecino M_j con la correspondiente medición del sensor relativo local, $(\rho, \varphi)_{ij}$. Esta correspondencia se realiza mediante el sensor relativo (cámara, lector de etiquetas RFID, etc.), o bien comparando la información de la postura del robot R_j (identidad M_j) recibida por comunicación con la postura determinada por el sensor relativo de R_i, aunque este último método puede resultar menos preciso.

- En este marco no se ha considerado que puedan ocurrir errores de identificación (*Matching*) ya que se asume que $\forall R_i$ es capaz de diferenciar entre un miembro del equipo G_R y un obstáculo estático o dinámico en el entorno.

- La arbitrariedad de D_r puede generar situaciones como por ejemplo que un robot R_i con D_r alto detecte y se comunique con un robot vecino R_j de D_r bajo aunque R_j no pueda observar a R_i. Estos casos se detallan de acuerdo al modelo de medición relativa expuesto en la sección 6.1.3.

- La postura L_k es estimada por cada R_i haciendo uso de todas las mediciones disponibles (locales, globales y relativas) y empleando las comunicaciones para realizar el intercambio de información cooperativo con los robots detectados dentro de su D_r. El algoritmo de fusión selecciona qué mediciones debe realizar en cada instante k según el esquema basado en eventos.

- Un R_i navega por el escenario según sea misión específica asignada y no requiere detener su movimiento para comunicarse, realizar las mediciones relativas o bien ejecutar el algoritmo de localización cooperativa. Esto difiere de algunos métodos existentes como el expuesto en [35].
- Ningún integrante de G_R está limitado a mantener una formación rígida ni determinada, aunque pueden realizar formaciones temporales de subconjuntos o del grupo completo si se requiere, tal y como se verá en el capítulo 7.
- El entorno se considera dinámico (robots y obstáculos en movimiento), por lo que las frecuencias con las que se detectan los robots vecinos son variantes en el tiempo y desconocidas a priori.
- Cada integrante R_i cuenta con un módulo de evasión de obstáculos (por ejemplo un algoritmo Braitenberg [163, 100]) y un algoritmo de planificación de trayectorias que determina la ruta que debe seguir el robot (evitando obstáculos y/o colisiones [109, 80, 110, 44, 111, 94, 93]).
- La información global asociada a un mapa del entorno puede estar disponible o ser obtenida únicamente por los $R_i \in G_{RA}$ si fuera necesario. No se ha contemplado sin embargo, el uso de SLAM cooperativo como en [61, 39, 126] debido a las limitaciones computacionales de los robots.
- Para cada robot en G_R se determina un modelo de acuerdo a su locomoción con el fin de utilizarlo en el algoritmo de localización y navegación.

6.1.2 Red de comunicación entre holones

Todos los holones o robots forman parte de una red multi-agente basada en los middlewares de JADE-LEJOS y JADE-ROS. A este nivel de abstracción, un holón se corresponde con un robot, las comunicaciones entre procesos locales distribuidos no se gestionan mediante esta comunicación. La abstracción de esta comunicación permite hacer uso del lenguaje de comunicación FIPA-ACL y dotar a cada R_i de un sistema de gestión de mensajes que realiza el intercambio de información según los principios que se detallan a continuación:

- El término "*comunicaciones*.[en] el ámbito del problema de localización en un grupo de holones cooperativos, se refiere a la transmisión de la información requerida para la localización cooperativa entre los miembros $R_i \in G_R$.
- La información transmitida se define como i_T (denotado con el subíndice T). Para cada R_i se incluye la postura estimada L_T, la covarianza del error de estimación de la postura $P_{T,p}$, las mediciones relativas $M_{T,r}$ (sólo si el vecino se encuentra dentro de D_r), las velocidades en los ejes globales $(\dot{x}_T, \dot{y}_T, \dot{\theta}_T)$ y el tiempo requerido para realizar el envío de la información (si existiera,

se determina mediante un temporizador activado al recibir una solicitud de información de un robot vecino). La información recibida i_R contempla las mismas variables (denotadas con el subíndice R) e incluye $M_{R,r}$ según el D_r del vecino.

- Los retardos en la trasmisión de la información causados por el medio de comunicación son conocidos y medibles, al conocer la distancia entre miembros y la velocidad de transmisión, o bien al utilizar un temporizador activado al realizar la llamada a un vecino y detenido al recibir su respuesta. Como alternativa para determinar el retardo puede utilizarse un módulo con reloj de tiempo real, si está disponible $\forall R_i \in G_R$ y se sincroniza de forma adecuada.

- Se supone para cualquier caso que $C_r \geq D_r$ pero cada R_i iniciará la comunicación únicamente con los vecinos contenidos en D_r por lo que si un robot puede detectar a un vecino, siempre podrá establecer comunicaciones con él. El número total de R_i contenidos en D_r se denotará como N_D.

- Cuando un R_i se comunica con un vecino, transmite su información i_T y espera la recepción de i_R. Esto se gestiona mediante múltiples hilos de ejecución en la unidad de control de la plataforma, que previenen cualquier espera bloqueante o interbloqueo mientras se recibe el mensaje de respuesta de la comunicación (se consideran comunicaciones no bloqueantes en cada R_i).

- El algoritmo de localización cooperativa (en adelante denotado como *CLA*) determina cuándo y con qué frecuencia el robot R_i se comunica con los vecinos contenidos en D_r. De esta forma, las comunicaciones en G_R no se llevan a cabo en cada instante de muestreo k (definido por el tiempo de muestreo T_s constante) o con una frecuencia de comunicación constante predefinida, siendo ésta variante en el tiempo.

- El *CLA* es distribuido, se ejecuta de forma local $\forall R_i \in G_R$, e inicia las comunicaciones cuando un evento R_A, definido según $P_{k,p}$ (elipsoides 3σ, ecuación (5.7)) supera un nivel predeterminado $R_{A,lim}$. Hasta que el evento no supera $R_{A,lim}$, R_i no inicia las comunicaciones con sus vecinos aunque estos se encuentren dentro del rango de detección D_r. Sin embargo, si un vecino solicita información (debido a que su R_A local superó su $R_{A,lim}$), el robot sí que enviará su i_T aunque su $R_{A,lim}$ no haya sido superado.

- La arbitrariedad de D_r entre los miembros puede ocasionar situaciones de ayuda desinteresada. Por ejemplo, un R_i puede enviar i_T para "*ayudar*.ª un vecino R_j si es necesario, aunque el sensor relativo de R_j no haya detectado a R_i (con lo que tampoco ha solicitado la información i_T). Esto es necesario cuando existen miembros del grupo con un D_r muy limitado, lo que causa que el robot no detecte rápidamente a un vecino para ejecutar la localización

cooperativa, aunque se haya alcanzado el $R_{A,lim}$, con lo que la covarianza del error de estimación puede crecer considerablemente. Por el contrario, si este robot ha superado su $R_{A,lim}$ y es detectado por un vecino con D_r alto y que envía la i_T aunque no sea solicitada, se podrá utilizar esta información para mejorar la estimación de la postura, utilizando el modelo de medición relativo de la sección 6.1.3, evitando así el crecimiento excesivo del error en la localización. Esta metodología de ayuda "*altruista*"puede administrarse dentro de cada robot R_i con rango D_r amplio, al comprobar si al enviar i_T se recibe o no del vecino R_j un i_R que incluye M_r (lo cual indica que el vecino R_j puede detectar al robot R_i con su sensor relativo). En caso de no recibir M_r, se agrega el identificador del vecino R_j a una lista (en R_i), según la cual el robot R_i le enviará i_T cada vez que detecte a R_j y aunque en R_i no se haya alcanzado el $R_{A,lim}$. Este método, a pesar de incrementar la cantidad de mensajes enviados en G_R, puede mejorar la precisión de la postura estimada por los robots con D_r muy limitado. También cabe destacar que el robot R_j (al que se le ayuda) puede decidir si utilizar o no la información i_R según la definición del evento dentro de este robot (si se ha alcanzado o no su $R_{A,lim}$).

- Además de la definición del evento R_A basada en el área de los elipsoides 3σ (ecuación (5.7)), es posible establecer definiciones adicionales según las necesidades específicas de la misión del grupo. Por ejemplo, puede definirse el evento de forma que se ejecute la fusión con la información relativa siempre que se detecte un nuevo robot vecino dentro de D_r (en cuyo caso se contactaría una sola vez hasta que no salga de D_r y vuelva a ser detectado), o bien siempre que N_D sea mayor a un valor mínimo $N_{D,min}$ (por ejemplo, ejecutar la fusión relativa si hay 3 o más vecinos dentro de D_r, $N_{D,min} = 3$, $N_D \geq N_{D,min}$). Es posible también, utilizar la información de la evitación de obstáculos para ejecutar la fusión relativa siempre que se evite un obstáculo y mejorar la precisión cuando se retoma la trayectoria deseada en el robot, o bien ejecutar el método cooperativo si el robot ha avanzado una distancia mínima (definida según la precisión de la estimación local del robot, obtenida de P_k). La selección del evento y de su valor límite se pueden definir de acuerdo a las necesidades propias de cada R_i a la hora de implementar el método.

- Cuando un R_i detecta múltiples vecinos dentro de su D_r (por ejemplo $N_D > 3$), el algoritmo cooperativo puede determinar y escoger qué medidas utilizar en la fusión sensorial basándose en la covarianza del error de estimación recibida $P_{R,p}$ y la medición relativa M_r (recibida $M_{R,r}$ o propia según disponibilidad). Esto permite al algoritmo *CLA* descartar la información que no es lo suficientemente precisa, basándose en los valores de la covarianza (mayor covarianza $\Rightarrow$ mayor incertidumbre $\Rightarrow$ menor precisión en la estimación de la postura), lo cual puede mejorar la precisión del método de localización además de disminuir el uso de recursos computacionales.

- Es posible realizar la etapa de corrección relativa de forma secuencial, calculando la etapa de corrección del filtro con dimensión unitaria (para un único vecino) y ejecutándola secuencialmente para cada vecino dentro de D_r. Esto puede disminuir el coste computacional al utilizar ecuaciones de menor dimensión además de facilitar la implementación del filtro cooperativo, esto se detalla en la descripción de *CLA* de la sección 6.2.

- Para poder llevar un registro y un histórico de lo ocurrido, es esencial proveer a cada R_i de la capacidad de almacenar localmente cada mensaje enviado y recibido, junto con el tiempo (local o del módulo de tiempo real) en el que se recibe cada i_R o se envía i_T, así como la identidad del robot al que se envía o del que se recibe la información.

Una vez definido el grupo multi-agente de robots y las reglas de intercambio de mensajes, se describe a continuación el modelo de medición relativa requerido por el algoritmo *CLA*.

6.1.3 Modelo de Medición Relativa

Existen en la literatura diversos modelos de medición relativa [22, 112, 113, 140, 55, 66, 139] definidos de acuerdo al tipo de información que proporciona un sensor relativo (rango, ángulo, rango y ángulo, etc.). Siguiendo el enfoque propuesto en estos trabajos, se define el modelo relativo utilizando el caso de dos robots vecinos con postura $\{L_1(x_1, y_1, \theta_1), L_2(x_2, y_2, \theta_2)\}$. Las mediciones relativas $M_r = \{\rho_{ij}, \varphi_{ij}\}$ se utilizan junto con la información i_R obtenida de los vecinos en D_r para obtener la medición de la postura, utilizando las relaciones geométricas entre M_r y cada L_R recibido, tal y como se muestra en la figura 6.1.

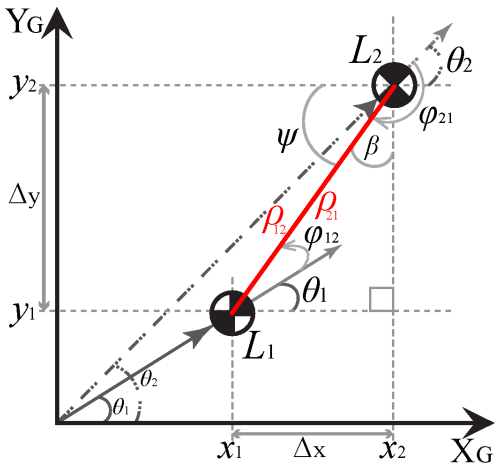


Figura 6.1: Descripción geométrica de la medición relativa M_r

Según muestra la figura 6.1, el robot R_1 ubicado en la posición L_1 mide el rango ρ_{12} y el ángulo relativo φ_{12}. De la misma forma el robot R_2 ubicado en L_2 obtiene ρ_{21} y φ_{21}. El ángulo relativo $[-\pi < \varphi \leq \pi]$ se mide en los ejes locales, por lo que desde los ejes globales se mide desde el ángulo actual de avance θ; si $\varphi = 0$ implica que el vecino se encuentra justo al frente del robot. La relación entre las posturas L_1 y L_2 con las distancias y ángulos relativos se muestra en la ecuación (6.1).

$$\begin{aligned} \rho_{12} &= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} = \sqrt{(\Delta x)^2 + (\Delta y)^2} \\ \psi &= \varphi_{12} + \theta_1 = \operatorname{atan2}(\Delta y / \Delta x) \end{aligned} \tag{6.1}$$

Puede utilizarse directamente el modelo de la ecuación (6.1) en los esquemas de fusión sensorial al utilizar su linealización, la cual aproxima la solución, o bien puede resolverse de forma exacta según el número N_D de vecinos R_j, D_r y la información i_R.

Se pueden emplear técnicas similares a la triangulación o trilateración de balizas con posición conocida [22, 59, 55, 78, 137] para obtener la postura del robot desde la información relativa cuando se detecta más de un R_j. Sin embargo, con el fin de obtener un método más general, éstos métodos *no* se han considerado en el presente trabajo, sino que se estudia el peor caso, el cual se produce cuando se detecta un único vecino R_j; ya que, como D_r puede ser distinto en cada robot, podría disponerse de menos información para determinar la postura de R_i.

En caso de que se detectara más de un vecino en D_r ($N_D > 1$), se puede utilizar la ecuación (6.1) N_D veces dentro del esquema de fusión, ajustando correspondientemente el vector de medición z_k y la matriz covarianza de la medición R_k para incorporar la información relativa de todos los vecinos detectados (bajo un esquema de corrección continua), o bien, puede incorporarse la ecuación (6.1) una única vez y realizar la actualización con la información relativa de forma secuencial (esquema de corrección por eventos).

De este modo, cuando $N_D = 1$, se establecen tres casos básicos utilizando el ejemplo mostrado en la figura 6.1 para obtener la postura de R_1 a partir de la información recibida de R_2 y la medición relativa según los valores de D_r:

1. R_1 y R_2 se detectan mutuamente: En este caso, ambos robots se encuentra en rango cuando $D_1 \cong D_2$ y la posición de cada R_i permite la detección (en cuyo caso $\rho_{12} \leq \{D_1 \cong D_2\}$. Para este caso R_1 recibe $\{L_2, \rho_{21}, \varphi_{21}\}$ y se

utilizan para obtener la postura completa L_1 según se muestra en la ecuación (6.2).

$$\begin{aligned} &\psi = \pi - \varphi_{21} + \theta_2, \quad \theta_1 = \psi - \varphi_{12} \Rightarrow \psi = \varphi_{12} + \theta_1 \\ &x_1 = x_2 - \rho_{12}\cos\psi \qquad\qquad y_1 = y_2 - \rho_{12}\sin\psi \end{aligned} \tag{6.2}$$

2. Sólo R_1 detecta a R_2: En este caso $D_1 > D_2$ y $\rho_{12} > D_2$. R_1 recibe únicamente $\{L_2\}$ y se utiliza para obtener los términos de la posición (x_1, y_1) en la postura L_1 pero no θ_1 según se muestra en la ecuación (6.3).

$$\begin{aligned} &\psi = \varphi_{12} + \theta_1, \\ &x_1 = x_2 - \rho_{12}\cos\psi \quad y_1 = y_2 - \rho_{12}\sin\psi \end{aligned} \tag{6.3}$$

3. Sólo R_2 detecta a R_1: En este caso $D_1 < D_2$ y $D_1 < \rho_{21}$. Si R_2 se comunica con R_1 para *ayudarle*, entonces R_1 recibe únicamente $\{L_2, \rho_{21}, \varphi_{21}\}$ pero no se dispondrá de $\{\rho_{12}, \varphi_{12}\}$. Al igual que ocurre en el caso anterior, con esta información únicamente se pueden determinar (x_1, y_1) en la postura L_1 pero no θ_1 según se muestra en la ecuación (6.4).

$$\begin{aligned} &\psi = \pi - \varphi_{21} + \theta_2, \\ &x_1 = x_2 - \rho_{21}\cos\psi \quad y_1 = y_2 - \rho_{21}\sin\psi \end{aligned} \tag{6.4}$$

Expuestos los distintos casos, las ecuaciones (6.2) a (6.4) son las que permiten incorporar la información relativa dentro del algoritmo de localización cooperativa tal y como se describe en la siguiente sección.

6.2 Algoritmos de Localización Distribuida Cooperativa: TCLA y ECLA

Con el objetivo de poder comparar una estrategia basada en eventos con una estrategia basada en una corrección continua, se desarrollan los dos algoritmos. El algoritmo de corrección continua se ha denominado *TCLA* y utiliza la información de la medida relativa en cada instante k junto con todas las mediciones locales y globales disponibles en la plataforma. Y el algoritmo de corrección por eventos

ECLA el cual sólo actualiza o corrige su posición cuando él considera que es necesario, reduciendo así el ancho de banda de las comunicaciones.

Los algoritmos están definidos para una plataforma de configuración diferencial cuyo modelo se establece en la ecuación (B.24) junto con las aceleraciones del modelo (B.34) (versión 3 partículas). Cabe destacar que los algoritmos son generales y pueden ser adaptados a otras plataformas utilizando los modelos correspondientes y ajustando las dimensiones de los vectores de estado x_k, entrada u_k y medición z_k según se requiera.

Para simplificar la descripción y notación del método cooperativo, a partir de ahora se exponen los algoritmos considerando que las mediciones de la velocidad $z_{k,v}$ y de la postura global $z_{k,p}$ (en caso de que el robot tenga acceso al sensor global), se incorporan mediante un EKF (predicción y corrección) ejecutado en cada instante k (basado en el tiempo, tradicional), siendo este EKF intercambiable por otros métodos expuestos en este capítulo según las capacidades computacionales de cada miembro del grupo.

Si, por ejemplo, un robot pertenece a G_{RA} es posible que realice la fusión mediante los algoritmos tradicionales (Algoritmos 1 y 2) o con asignación de entradas (Algoritmos 3 y 4) suponiendo que no existen problemas de disponibilidad en la medición, pero también es posible que ejecute un algoritmo basado en eventos (algoritmo 11) de manera que se limite el acceso al sensor global y se tomen en cuenta los retardos temporales asociados, o utilizando un nivel del evento $R_{A,lim}$ bajo, para acceder a la información global más frecuentemente que un robot de recursos limitados. Sin embargo, para los robots pertenecientes a G_{RL}, se puede recurrir a los algoritmos en cascada reducido (Algoritmo 5) en caso de no tener problemas de acceso al sensor global, o al algoritmo en cascada con el modelo en cascada (algoritmo 7), el cual puede utilizarse además sin la corrección global en caso de no disponer del sensor global. De igual forma, pueden utilizarse los algoritmos en cascada por eventos (algoritmos 10 y 11) si se debe limitar el acceso al sensor global, en cuyo caso el $R_{A,lim}$ será mayor que para el caso de los integrantes de G_{RA}, con el fin de disminuir el acceso al sensor global y utilizarlo sólo en caso necesario.

Por estas razones, en los algoritmos del presente capítulo se describe principalmente la etapa de corrección de la estimación mediante la información cooperativa.

La ecuación (6.5) expone las definiciones utilizadas para el filtro de fusión, x_k, u_k y z_k, las cuales se corresponden a las mismas de la sección anterior pero agregando la medición del sensor relativo $z_{k,r}$ obtenida de la medición relativa M_r y la

información recibida mediante comunicación i_R según las ecuaciones del modelo relativo (ecuaciones de la (6.1) a la (6.4)).

$$\begin{aligned}
\boldsymbol{x}_{k,p} &= \begin{bmatrix} x & y & \theta \end{bmatrix}_k^T, \quad \boldsymbol{x}_{k,v} = \begin{bmatrix} v_x & \omega \end{bmatrix}_k^T \\
\boldsymbol{x}_k &= \begin{bmatrix} x_{k,p} & x_{k,v} \end{bmatrix}^T \\
\boldsymbol{u}_k &= \begin{bmatrix} a & \alpha \end{bmatrix}_k^T \\
\boldsymbol{z}_{k,p} &= H_{k,p} x_{k,p} = \boldsymbol{L}_{GM} = \begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}_{GM}^T \\
\boldsymbol{z}_{k,v} &= H_{k,v} x_{k,v} = \begin{bmatrix} v_{x,enc} & \omega_{enc} & \omega_{gyr} & \omega_{comp} \end{bmatrix}_k^T \\
\boldsymbol{z}_{k,r} &= H_{k,r} x_{k,p} = \begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}_r^T
\end{aligned} \tag{6.5}$$

6.2.1 Algoritmo de Localización Cooperativa con Corrección Continua

La primera aproximación al problema se aborda con el desarrollo de un algoritmo de localización cooperativa con corrección continua que se ha denominado **TCLA**. Este algoritmo utiliza la información relativa en cada periodo de muestreo k junto con todas las mediciones locales y globales disponibles en la plataforma $z_k = [z_{k,p} \quad z_{k,v}]$. Mediante la definición de x_k, u_k y z_k de la ecuación (6.5) y junto con el modelo del robot diferencial (ecuaciones (B.34) y (B.24)) se constituye el EKF cooperativo con corrección continua mostrado en el algoritmo 12, el cual utiliza las covarianzas del proceso Q_k y medición R_k para realizar la estimación del estado x_k y la covarianza del error de estimación P_k.

Esta metodología responde a una aproximación "tradicional" de los métodos cooperativos mediante la utilización del modelo relativo (6.1), ya que se asigna la ecuación (6.1) $\forall R_j$ dentro de cada D_r detectado, por lo que si $N_D \geq 1$ se deberán ajustar dinámicamente las dimensiones de z_k, R_k y H_k en cada instante k, de acuerdo a N_D. Hecho esto, se incorpora cada M_r realizado y cada i_R recibido en la etapa de corrección de la postura con la información relativa. Con la identificación o "matching" resuelto, cada ecuación (6.1) agregada se utiliza con el correspondiente M_r e i_R de cada vecino identificado.

El algoritmo 12 refleja en pseudocódigo el funcionamiento del *TCLA*. Como puede observarse, este método tiene toda la información necesaria para su interpretación disponible en cada instante k, lo cual requiere en cada iteración llevar a cabo una inversión de matrices normalmente de grandes dimensiones (según z_k y N_D), por lo que requiere un uso extenso del procesador y de la memoria de la plataformano, no resultando de esta manera adecuado para los robots de recursos limitados pertenecientes al grupo definido como G_{LR}. Además, el método requiere también

un uso considerable del ancho de banda ya que cada robot debe comunicarse con todos los vecinos en D_r para obtener i_R cada instante k.

Algoritmo 12: Algoritmo EKF cooperativo, distribuido y recursivo, corrección basada en el tiempo: *TCLA*

Entrada: u_{k-1}=$[\, a \;\; \alpha \,]^T$,x_{k-1},P_{k-1}
Medición: $z_{k,p}$,$z_{k,v}$,M_r
Salida: $\hat{x}_k$,P_k
Datos: $f(\cdot)$ *y* $h(\cdot)$ *del modelo* (B.24),Q_k,R_k,i_R
Inicialización: x_0,P_0
Para *el instante actual* k **hacer**
  Predicción:
    $\hat{x}_k = f\left(\hat{x}_{k-1}, u_{k-1}, 0\right)$
    $A_k = \frac{\partial f}{\partial x}\Big|_{\hat{x}_{k-1},u_{k-1},0}$, $W_k = \frac{\partial f}{\partial w}\Big|_{\hat{x}_{k-1},u_{k-1},0}$
    $P_k = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$
  Actualización Cooperativa:
  Si $N_D \geq 1$ **entonces**
    $M_r = [\rho_{ij} \quad \varphi_{ij}]^T$, $j = 1 \cdots N_D$
    $z_k = \left[\begin{array}{cc} z_{k,p} & z_{k,v} \quad M_r \end{array} \right]^T$
    Ajustar dimensión de R_k (según M_r y N_D)
    Agregar la ecuación (6.1) a $\boldsymbol{h()}$ (N_D veces)
  si no
    $z_k = \left[\begin{array}{cc} z_{k,p} & z_{k,v} \end{array} \right]^T$
    Utilizar R_k con $N_D = 0$
    Utilizar $\boldsymbol{h()}$ sin la ecuación (6.1)
  fin
  H_k=$\frac{\partial h}{\partial x}\big|_{\hat{x}_k,0}$,$N_k = \frac{\partial h}{\partial n}\big|_{\hat{x}_k,0}$
  Corrección:
    $K_k = P_k H_k^T \left(H_k P_k H_k^T + N_k R_k N_k^T\right)^{-1}$
    $\hat{x}_k = \hat{x}_k + K_k \left[z_{k,r} - h\left(\hat{x}_k, 0\right)\right]$
    $P_k = \left(I - K_k H_k\right) P_k$
fin

6.2.2 Algoritmo de Localización Cooperativa con Corrección basada en Eventos

Del mismo modo que se orientó la mejora del algoritmo de autolocalización global definido en el capítulo 5, el algoritmo de localización cooperativa basado en eventos (**ECLA**) obtiene y utiliza la información relativa detectada para actualizar la estimación de la postura únicamente cuando un evento R_A supera su valor límite predefinido $R_{A,lim}$. De este modo se reduce el procesamiento, la memoria y el ancho de banda requerido por la estrategia de corrección continua *TCLA* haciendo plausible su implementación incluso en los robots con menores recursos disponibles. Del mismo modo que ocurría en el algoritmo 11, el valor del evento refleja la consciencia de mal posicionamiento e indicará cuando el error local es considerable, definiendo $R_{A,lim}$ según la covarianza de la estimación de la postura, calculada en cada instante k por el filtro aplicado.

Partiendo del algoritmo 12 desarrollado, es decir, el *TCLA*, es necesaria la supresión de la inversión y manipulación de matrices de grandes dimensiones (en función de N_D) como primer paso para incorporar la estrategia basada en eventos. Para ello se propone actualizar la estimación de la postura con la información relativa de forma secuencial, al considerar la fusión para un único vecino pero realizando la etapa de corrección N_D veces. De este modo se minimizan los requerimientos en procesamiento y memoria (inversa de menor dimensión) aunque aun puede requerirse un tiempo de muestreo considerablemente alto según las características de la plataforma y según el valor máximo de N_D.

Para seguir reduciendo costes, se considera la medición relativa como un sensor de postura adicional $z_{k,r} = \begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}_r^T$. La actualización de la estimación de la postura se realiza incorporando $z_{k,r}$ mediante una etapa de corrección KF en cascada, similar a la del algoritmo 5 (cascada reducido), la cual se ejecuta recursivamente sólo si $N_D > 1$. El cálculo de $z_{k,r}$ se realiza de acuerdo al valor de D_r, utilizando los correspondientes M_r e i_R según los casos expuestos en las ecuaciones (6.2) a (6.4).

Al utilizar $z_{k,r}$ en la fusión sensorial, es necesario realizar una propagación de la covarianza asociada a M_r e i_r según el caso de D_r a través del modelo relativo, con el fin de obtener la $R_{k,r}$ necesaria a utilizar en el filtro de fusión, para lo cual se procede con una estrategia similar a la utilizada en los filtros en cascada reducido (ecuación (3.25)).

Se realiza entonces una aproximación lineal de $R_{k,r}$ mediante la ecuación (6.6), donde $\nabla \boldsymbol{z}_u$ es el operador gradiente aplicado a las ecuaciones (6.2) y a (6.4)) respecto a las *entradas* del modelo relativo (las cuales pueden incluir $x_2, y_2, \theta_2, \rho_{12}, \varphi_{12}, \varphi_{21}$), y R_{M_r,i_R} es la covarianza asociada a M_r e i_r según D_r. Cabe nombrar que el operador gradiente realiza la linealización del modelo relativo (según D_r) respecto a las entradas del mismo, correspondiente a las variables de M_r e i_r, lo cual per-

mite propagar la covarianza de la entrada del modelo relativo u^* a la salida del mismo y^* y obtener $R_{k,r}$. Esto ocurre de forma similar a la ecuación (3.25), con la diferencia de que para $R_{k,r}$ la propagación en el tiempo no se realiza mediante la aproximación lineal (ecuación (6.6)), sino que se considera que ésta es realizada en las covarianzas asociadas a las propias del sensor relativo, M_r, y a las recibidas mediante comunicación, i_r.

$$R_{k,r} = R_{x_1 y_1 \theta_1} = \nabla z_u R_{M_r, i_R} \nabla z_u^T \tag{6.6}$$

Aplicando la ecuación (6.6) a las ecuaciones (6.2), (6.3) y (6.4), se obtiene $R_{k,r}$ para los distintos casos de detección tal y como se muestra en las ecuaciones (6.7), (6.8) y (6.9), donde se considera una matriz R_{M_r,i_R} diagonal.

Sobre la ecuación (6.2)

$$u^* = \begin{bmatrix} x_2 & y_2 & \theta_2 & \rho_{12} & \varphi_{12} & \varphi_{21} \end{bmatrix}^T, y^* = \begin{bmatrix} x_1 & y_1 & \theta_1 \end{bmatrix}^T$$

$$\nabla z_u = \begin{bmatrix} 1 & 0 & 0 & -\cos(\varphi_{12}+\theta_1) & \rho_{12}\sin(\varphi_{12}+\theta_1) & 0 \\ 0 & 1 & 0 & -\sin(\varphi_{12}+\theta_1) & -\rho_{12}\cos(\varphi_{12}+\theta_1) & 0 \\ 0 & 0 & 1 & 0 & -1 & -1 \end{bmatrix}$$

$$R_{M_r,i_R} = \begin{bmatrix} \sigma^2_{x_2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{y_2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2_{\theta_2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma^2_{\rho_{12}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{\varphi_{12}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma^2_{\varphi_{21}} \end{bmatrix} = \begin{bmatrix} r_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & r_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & r_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & r_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & r_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & r_{66} \end{bmatrix} \tag{6.7}$$

$$R_{k,r} = R_{x_1 y_1 \theta_1} = \nabla z_u R_{M_r,i_R} \nabla z_u^T = \begin{bmatrix} r_{1a} & r_{1b} & r_{1c} \\ r_{1d} & r_{1e} & r_{1f} \\ r_{1g} & r_{1h} & r_{1i} \end{bmatrix}$$

$$\begin{aligned} r_{1a} &= r_{55}\rho_{12}^2 \sin^2(\varphi_{12}+\theta_1) + r_{44}\cos^2(\varphi_{12}+\theta_1) + r_{11} \\ r_{1b} = r_{1d} &= -r_{55}\rho_{12}^2 \cos(\varphi_{12}+\theta_1)\sin(\varphi_{12}+\theta_1) \\ &\quad + r_{44}\cos(\varphi_{12}+\theta_1)\sin(\varphi_{12}+\theta_1) \\ r_{1c} = r_{1g} &= -r_{55}\rho_{12}\sin(\varphi_{12}+\theta_1) \\ r_{1e} &= r_{55}\rho_{12}^2 \cos^2(\varphi_{12}+\theta_1) + r_{44}\sin^2(\varphi_{12}+\theta_1) + r_{22} \\ r_{1f} = r_{1h} &= r_{55}\rho_{12}\cos(\varphi_{12}+\theta_1) \\ r_{1i} &= r_{33} + r_{55} + r_{66} \end{aligned}$$

Sobre la ecuación (6.3)

$$u^* = \begin{bmatrix} x_2 & y_2 & \rho_{12} & \varphi_{12} \end{bmatrix}^T, y^* = \begin{bmatrix} x_1 & y_1 & \theta_1 \end{bmatrix}^T$$

$$\nabla z_u = \begin{bmatrix} 1 & 0 & -\cos(\varphi_{12}+\theta_1) & \rho_{12}\sin(\varphi_{12}+\theta_1) \\ 0 & 1 & -\sin(\varphi_{12}+\theta_1) & -\rho_{12}\cos(\varphi_{12}+\theta_1) \end{bmatrix}$$

$$R_{M_r,i_R} = \begin{bmatrix} \sigma^2_{x_2} & 0 & 0 & 0 \\ 0 & \sigma^2_{y_2} & 0 & 0 \\ 0 & 0 & \sigma^2_{\rho_{12}} & 0 \\ 0 & 0 & 0 & \sigma^2_{\varphi_{12}} \end{bmatrix} = \begin{bmatrix} r_{11} & 0 & 0 & 0 \\ 0 & r_{22} & 0 & 0 \\ 0 & 0 & r_{33} & 0 \\ 0 & 0 & 0 & r_{44} \end{bmatrix} \tag{6.8}$$

$$R_{k,r} = R_{x_1 y_1 \theta_1} = \nabla z_u R_{M_r,i_R} \nabla z_u^T = \begin{bmatrix} r_{1a} & r_{1b} \\ r_{1c} & r_{1d} \end{bmatrix}$$

$$r_{1a} = r_{44}\rho_{12}^2 \sin^2(\varphi_{12}+\theta_1) + r_{33}\cos^2(\varphi_{12}+\theta_1) + r_{11}$$

$$\begin{aligned} r_{1b} = r_{1c} = &-r_{44}\rho_{12}^2 \cos(\varphi_{12}+\theta_1)\sin(\varphi_{12}+\theta_1) \\ &+ r_{33}\cos(\varphi_{12}+\theta_1)\sin(\varphi_{12}+\theta_1) \end{aligned}$$

$$r_{1d} = r_{44}\rho_{12}^2 \cos^2(\varphi_{12}+\theta_1) + r_{33}\sin^2(\varphi_{12}+\theta_1) + r_{22}$$

Sobre la ecuación (6.4)

$$u^* = \begin{bmatrix} x_2 & y_2 & \theta_2 & \rho_{21} & \varphi_{21} \end{bmatrix}^T, y^* = \begin{bmatrix} x_1 & y_1 & \theta_1 \end{bmatrix}^T$$
$$\psi = \pi - \varphi_{21} + \theta_2$$
$$\nabla z_u = \begin{bmatrix} 1 & 0 & \rho_{21}\sin(\psi) & -\cos(\psi) & -\rho_{21}\sin(\psi) \\ 0 & 1 & -\rho_{21}\cos(\psi) & -\sin(\psi) & \rho_{21}\cos(\psi) \end{bmatrix}$$

$$R_{M_r,i_R} = \begin{bmatrix} \sigma^2_{x_2} & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{y_2} & 0 & 0 & 0 \\ 0 & 0 & \sigma^2_{\theta_2} & 0 & 0 \\ 0 & 0 & 0 & \sigma^2_{\rho_{21}} & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{\varphi_{21}} \end{bmatrix} = \begin{bmatrix} r_{11} & 0 & 0 & 0 & 0 \\ 0 & r_{22} & 0 & 0 & 0 \\ 0 & 0 & r_{33} & 0 & 0 \\ 0 & 0 & 0 & r_{44} & 0 \\ 0 & 0 & 0 & 0 & r_{55} \end{bmatrix} \quad (6.9)$$
$$R_{k,r} = R_{x_1 y_1 \theta_1} = \nabla z_u R_{M_r,i_R} \nabla z_u^T = \begin{bmatrix} r_{1a} & r_{1b} \\ r_{1c} & r_{1d} \end{bmatrix}$$

$$\begin{aligned} r_{1a} &= r_{44}\cos^2(\varphi_{21} - \theta_2) + r_{33}\rho^2_{21}\sin^2(\varphi_{21} - \theta_2) \\ &\quad + r_{55}\rho^2_{21}\sin^2(\varphi_{21} - \theta_2) + r_{11} \\ r_{1b} &= r_{1c} = r_{33}\rho^2_{21}\cos(\varphi_{21} - \theta_2)\sin(\varphi_{21} - \theta_2) \\ &\quad - r_{44}\cos(\varphi_{21} - \theta_2)\sin(\varphi_{21} - \theta_2) \\ &\quad + r_{55}\rho^2_{21}\cos(\varphi_{21} - \theta_2)\sin(\varphi_{21} - \theta_2) \\ r_{1d} &= r_{44}\sin^2(\varphi_{21} - \theta_2) + r_{33}\rho^2_{21}\cos^2(\varphi_{21} - \theta_2) \\ &\quad + r_{55}\rho^2_{21}\cos^2(\varphi_{21} - \theta_2) + r_{22} \end{aligned}$$

Como puede observarse en las ecuaciones, $R_{k,r}$ es variante en el tiempo debido a que depende de los valores que se reciban en R_{i_R} para cada instante k, con lo que la ganancia del filtro que incorpora la medición relativa debe calcularse en cada instante k y N_D veces según la cantidad de vecinos detectados.

Mediante esta metodología, se incorpora la etapa de corrección basada en eventos al algoritmo, para lo cual se utiliza el evento definido en la sección 5.2.2, ecuación (5.7), con lo que se obtiene y utiliza la información relativa para mejorar la estimación de la postura del robot sólo si el evento R_A, definido según el área de los elipsoides 3σ (calculada a partir de $P_{k,p}$) normalizada con el área A_{Ri} del robot R_i, supera un nivel predeterminado definido como $R_{A,lim}$. Del mismo modo que ocurría para la autolocalización global por eventos, este límite se escoge como un compromiso entre el número de consultas realizadas entre robots (del que depende en uso del ancho de banda, consumo de energía y tiempo de cómputo) y la precisión mínima deseada de la postura estimada.

Algoritmo 13: Algoritmo EKF cooperativo, distribuido y recursivo, corrección basada en eventos: *ECLA*

Entrada: u_{k-1}=$[\,a\ \ \alpha\,]^T$,x_{k-1},P_{k-1}
Medición: $z_k = \left[\ z_{k,p} \quad z_{k,v}\ \right]^T$,$M_r$,$i_R$
Salida: $\hat{x}_k$,P_k
Datos: *f y h del modelo* (B.24),Q_k,R_k,A_{Ri},$R_{A,lim}$,$H_{k,r} = I_{3\times 3}$
Inicialización: x_0,P_0
Para *el instante actual k* **hacer**
  Predicción:
  $\hat{x}_k = f\left(\hat{x}_{k-1}, u_{k-1}, 0\right)$
  $A_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1},u_{k-1},0}$, $W_k = \left.\frac{\partial f}{\partial w}\right|_{\hat{x}_{k-1},u_{k-1},0}$, H_k=$\left.\frac{\partial h}{\partial x}\right|_{\hat{x}_k,0}$, $N_k = \left.\frac{\partial h}{\partial n}\right|_{\hat{x}_k,0}$
  $P_k = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$
  Corrección local y global:
  $K_k = P_k H_k^T \left(H_k P_k H_k^T + N_k R_k N_k^T\right)^{-1}$
  $\hat{x}_k = \hat{x}_k + K_k\left[z_k - h\left(\hat{x}_k, 0\right)\right]$
  $P_k = \left(I - K_k H_k\right) P_k$
  Actualización Cooperativa KF:
  Área elipsoide 3σ: A_{ellip} (ecuación (5.7))
  Calcular Evento: $\boldsymbol{R}_A = A_{ellip}/A_{Ri}$
  Si $\boldsymbol{R}_A > R_{A,lim}$ **entonces**
    Obtener M_r e i_R $\forall R_j$ en D_r
  fin
  Si *Se han obtenido* N_D *mediciones* M_r **entonces**
    Realizar Matching de cada M_r con cada i_R
    Opcional: Descartar las medidas de baja precisión o escoger la mejor
    Obtener $z_{k,r}$ $\forall R_j \in D_r$ según el caso de D_r: ecuaciones (6.2) a (6.4)
    Obtener $R_{k,r}$ $\forall R_j \in D_r$ según la ecuación (6.6) ((6.7) a (6.9))
    Ejecutar la corrección relativa (secuencialmente si $N_D > 1$):
      $K_{k,p} = P_{k,p} H_{k,r}^T \left(H_{k,r} P_{k,p} H_{k,r}^T + R_{k,r}\right)^{-1}$
      $\hat{x}_{k,p} = \hat{x}_{k-1,p} + K_{k,KF}\left(z_{k,r} - H_{k,r}\hat{x}_{k-1,p}\right)$
      $P_{k,p} = \left(I - K_{k,p} H_{k,r}\right) P_{k,p}$
  fin
fin

El algoritmo 13 muestra la estructura de la implementación de la metodología de localización cooperativa basada en eventos *ECLA*. El funcionamiento sigue la misma línea que los algoritmos basados en eventos para autolocalización, cuando se cumple la condición del evento ($R_A > R_{A,lim}$), el algoritmo corregirá la estimación de la postura $x_{k,p}$ (obtenida con la información local y global) mediante el uso de la medición relativa $z_{k,r}$ y la covarianza de medición $R_{k,r}$. La covarianza del error resultante $P_{k,p}$ decrecerá de acuerdo a $R_{k,r}$ cuando se actualice la postura, con lo que se restablecerá el valor del evento, volviéndose a cumplir $R_A < R_{A,lim}$, lo cual inhabilitará la corrección con la información relativa hasta que se vuelva a dar la condición del evento, es decir, cuando $R_A > R_{A,lim}$.

Cuanto menores sean los valores correspondientes a $R_{k,r}$, menor será la dimensión de la matriz $P_{k,p}$ y menor necesidad de cómputo tendrá el algoritmo. Por ejemplo, casos en los que un robot determina M_r mediante un sensor con una precisión adecuada y i_R se recibe de un vecino $R_j \in G_{RA}$ con acceso a la información global. O también, casos en los que i_R es obtenida de un vecino que ha actualizado recientemente su postura a partir de otro $R_j \in G_{RA}$ con acceso a la información global. De igual modo, si i_R proviene de un robot sin acceso a la información global (o perteneciente a G_{RL}) puede darse un decremento en $P_{k,p}$ lo cual disminuirá R_A levemente. Para este caso en particular se alcanzará $R_{A,lim}$ con mayor frecuencia hasta que se realice la actualización utilizando una i_R con $R_{k,r}$ baja.

Un robot que pertenece a G_{RA} puede definir dos $R_{A,lim}$ uno para la etapa global y otro para la localización cooperativa. Bajo esta premisa es posible priorizar uno de los sensores, es decir, utilizar un $R_{A,lim,p}$ bajo para el sensor global (GPS por ejemplo) de forma que se use frecuentemente y un $R_{A,lim,r}$ mayor en el caso del método cooperativo. De esta manera, el algoritmo utilizaría mayoritariamente el sensor global para corregir la postura y en caso de que por alguna razón se perdiera el acceso a éste, la covarianza crecería lo suficiente como para superar $R_{A,lim,r}$, con lo que se emplearía el método cooperativo para actualizar la postura y reducir así el error de estimación y su respectiva covarianza. También en el caso de los integrantes de G_{RA}, se puede considerar el caso de $R_{A,lim,p} = R_{A,lim,r}$ y asignar ambos con un nivel de evento bajo si no se requiere restringir en gran medida la utilización del ancho de banda o alto en el caso contrario. Este tipo de decisiones deben considerarse durante la implementación del grupo heterogéneo de robots y vienen definidas por las restricciones determinadas según las características de cada miembro.

En cambio, cuando un robot pertenece a G_{RL} es posible que, debido a sus limitaciones, tenga únicamente acceso a la localización cooperativa, en cuyo caso sólo se requiere la asignación de $R_{A,lim,r}$ según la necesidad de limitación del ancho de banda del robot, a valores bajos, mayor uso del ancho de banda. En caso de que el robot disponga de acceso a ambos sensores es posible priorizar alguno de ellos del mismo modo que ocurría para el grupo G_{RA} o también es posible asignar

$R_{A,lim,p} = R_{A,lim,r}$ a un nivel alto con el fin de restringir el uso del ancho de banda de la plataforma cuando se utilizan ambos sensores.

Analizadas las distintas posibilidades de la definición del $R_{A,lim}$, se puede afirmar que la implementación de la localización cooperativa ofrece un alto nivel de flexibilidad en los distintos integrantes de G_{RA} y G_{RL}. De esta forma y a diferencia de los métodos existentes, es posible utilizar el mismo algoritmo 13 distribuido para realizar la localización de grupos heterogéneos simplemente adaptando, según las capacidades de cada plataforma, la definición de su evento y de su límite.

En los casos en los que un robot obtiene una detección de vecinos con $N_D > 1$, es posible reducir recursos si en lugar de realizar una actualización secuencial con cada información relativa recibida (N_D actualizaciones), se selecciona un subconjunto de las mejores o, simplemente la mejor. Esto es posible dado que en la información relativa que se recibe cada vecino incluye la covarianza asociada a su postura por lo que el algoritmo tan sólo debe escoger el vecino con menor covarianza para actualizarse con el mejor posicionado, reduciendo así el uso de procesador y memoria. Este aspecto es interesante sobre todo para integrantes del grupo G_{LR} cuyas prestaciones son más reducidas.

Por otro lado, cabe mencionar también la incapacidad de corrección de la orientación en los casos de detección 2 y 3 (ecuaciones (6.3) y (6.4)), los cuales serán los más frecuentes para miembros de G_{LR} por tener un rango de detección reducido. Esto puede generar que la covarianza del error de la orientación θ se incremente indefinidamente por lo que es recomendable que estos robots incorporen algún sensor de orientación absoluta como una brújula por ejemplo, para evitar este problema. Otra idea aplicable para evitar esta situación es incorporar algún método de triangulación cuando $N_D > 3$ para obtener la θ del robot en cuestión.

Teniendo en cuenta todas estas consideraciones, el método de localización cooperativo basado en eventos que se ha propuesto en el presente capítulo tiene la ventaja de ser flexible y aplicable a cualquier algoritmo de localización. Además, es posible seleccionar distintos tipos de evento para realizar la corrección cooperativa según los requerimientos de cada $R_i \in G_R$, permitiendo contemplar distintos casos de $R_{A,lim}$ y D_r.

Una vez definido el método cooperativo se procede con la exposición de los resultados obtenidos en las pruebas empíricas realizadas.

6.3 Resultados Obtenidos en Pruebas Empíricas

Cuando se trabaja con múltiples plataformas reales compartiendo un mismo escenario, no resulta sencillo obtener una ejecución libre de imprevistos que puedan alterar los resultados. Además, los algoritmos de localización cooperativa exigen que los robots sean capaces de detectarse entre sí mediante algún sensor embarcado para este fin, lo cual excluye por ejemplo plataformas como LEGO NXT o e-puck ya que no cumplen con el mínimo de especificaciones requerido para ello. Por este motivo las pruebas empíricas se dividieron en dos fases. Una primera fase donde se implementó una plataforma de simulación desarrollada expresamente para esta tesis y una segunda fase donde se realizó un experimento con robots reales de configuración diferencial modelo Summit XL.

6.3.1 Plataforma o Monitor de Simulación

Además de las plataformas experimentales, durante el desarrollo de esta tesis surgió la necesidad de disponer de una plataforma o monitor de simulación donde poder trabajar con un escenario compartido por múltiples robots sin lidiar con los imprevistos añadidos de trabajar con robots reales.

La plataforma se ha implementado como un módulo independiente más del sistema multi-agente distribuido basado en JAVA y compatible con los middlewares desarrollados JADE-LeJOS y JADE-ROS.

La interfaz se ejecuta y se administra a partir de un agente creado específicamente para monitorizar el sistema distribuido. Este agente se encarga de suscribirse a los topics de posición de todos los robots y representar su posición dentro del escenario.

Para simular los robots, se han definido agentes que implementan el modelo dinámico de las aceleraciones de las ruedas del robot de la ecuación (B.57) que se utiliza como entrada al modelo (B.34) (versión 3 partículas) el cual define las entradas del modelo de la postura del robot diferencial de la ecuación (B.24). De este modo, el agente software ejecuta una réplica del algoritmo de control del robot real y, migrar o integrar la plataforma de simulación con robots reales, tan sólo requiere el traslado del código entre plataformas. Los agentes robóticos simulados también publican su posición en cada iteración del bucle de control y es el agente encargado de la plataforma quien lee sus posiciones y otros datos de interés, y las actualiza visualmente en la interfaz, centralizando los datos de todos los agentes que intervienen.

Dicha centralización otorga el beneficio de poder registrar en ficheros de texto la evolución de información de una manera temporal ordenada para después poder analizarla. De otra manera, este ejercicio de sincronización resulta realmente complejo en sistemas desacoplados donde cada plataforma utiliza su propio reloj.

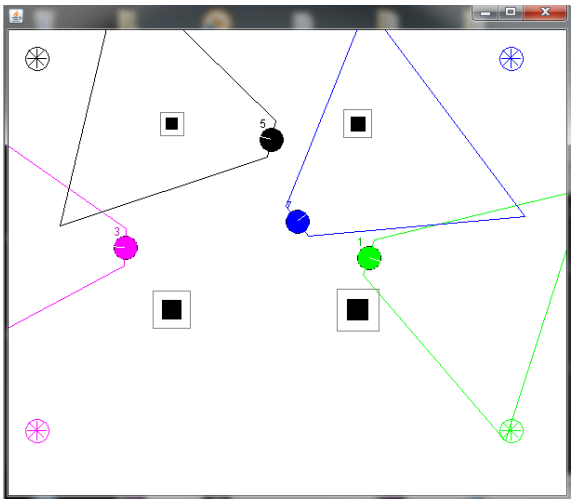


Figura 6.2: Monitor de simulación

La interfaz de la plataforma de simulación se observa en la figura 6.2, donde se define un entorno de navegación delimitado por las paredes que definen la ventana del escenario, de forma que el movimiento de los robots esté contenido. Cada R_i se muestra como un círculo pequeño con un color de relleno claro, en el cual se indica el correspondiente valor del índice i del robot y la dirección (orientación) de avance del mismo mediante una línea interna blanca (del centro hacia el borde) y normalmente su rango de detección en caso de tenerlo (representado por un trapecio en la figura 6.2).

La detección virtual de objetos utiliza dicha información centralizada para identificar posibles detecciones, así como las medidas relativas de dichas detecciones. Esto se ha implementado de manera que cada agente ejecuta un módulo de detección independiente el cual se suscribe a los datos publicados por el agente gestor de la plataforma (quien conoce toda la información centralizada).

Este monitor visual de sucesos entre agentes ha sufrido distintas modificaciones a lo largo del desarrollo de este trabajo. Se ha utilizado tanto en problemas de localización de robots como en problemas de navegación, coordinación y evasión de obstáculos (capítulo 8).

6.3.2 Resultados Obtenidos en Simulación

Se presentan dos pruebas relevantes para analizar tanto la evolución de la covarianza del error de estimación (del grupo de robots G_R y de cada miembro R_i de éste), como la cantidad de mensajes transmitidos dentro de G_R (enviados y recibidos entre sus integrantes) para la localización cooperativa. Además, se asigna en los algoritmos *TCLA* y *ECLA* un rango de detección D_r igual en cada R_i, así como la fusión cooperativa utilizando únicamente la mejor información relativa recibida (la de menor covarianza $R_{k,r}$ recibida). Esto permite realizar una mejor comparación entre la actualización temporal y por eventos, ya que la principal diferencia entre los algoritmos queda reducida al tipo de actualización y por lo tanto a la forma en la que se realiza el intercambio de mensajes (en cada instante k, o cuando el evento, basado en el área normalizada de los elipsoides 3σ, supera un nivel predeterminado: $R_A > R_{A,lim}$, ecuación (5.7)).

La primera prueba consiste en la simulación de un grupo de cinco integrantes R_i realizando un seguimiento de una trayectoria lineal cíclica, en la que los robots avanzan hacia el punto de destino y una vez alcanzado regresan al punto de partida, lo cual permite el estudio del esquema cooperativo cuando las frecuencias de detección de cada vecino son regulares o aproximadamente constantes.

La segunda prueba se realiza al considerar un grupo de diez integrantes que se mueven en el entorno con velocidad constante y ejecutan un algoritmo de evasión de obstáculos Braitenberg, evitando colisiones con los límites del del escenario marcado a modo de recinto cuadrado y con los robots vecinos. Esta configuración en particular permite observar el comportamiento del esquema cooperativo cuando las frecuencias de detección son variantes en el tiempo.

6.3.2.1 Grupo de Cinco Robots con Trayectorias Lineales Cíclicas

Esta prueba consiste en la simulación de un grupo G_R de cinco agentes los cuales realizan un seguimiento de una trayectoria lineal cíclica durante 20min. Para recrear la heterogeneidad del grupo, se decide que únicamente uno de los integrantes, R_1, tiene acceso a la medición global de su postura. De este modo, se considera que R_1 tiene un error de estimación de la postura nulo y por lo tanto, que la covarianza de su error de estimación es también nula. Esto facilita poder observar con más claridad el efecto de la corrección con información relativa y la diferencia entre utilizar la información de un vecino con baja covarianza (con acceso al sensor global o recientemente actualizado con la información de éste vecino), y el utilizar la información de un vecino de covarianza alta. En realidad, aunque el error de estimación y la covarianza son distintos de cero, suelen ser valores reducidos si el sensor global tiene una precisión adecuada. Además, en el esquema cooperativo al menos un robot debe poseer acceso a la información global, de forma que se

proporciona convergencia global al grupo según se expone en [150] (se evita el crecimiento no acotado del error de estimación de la postura).

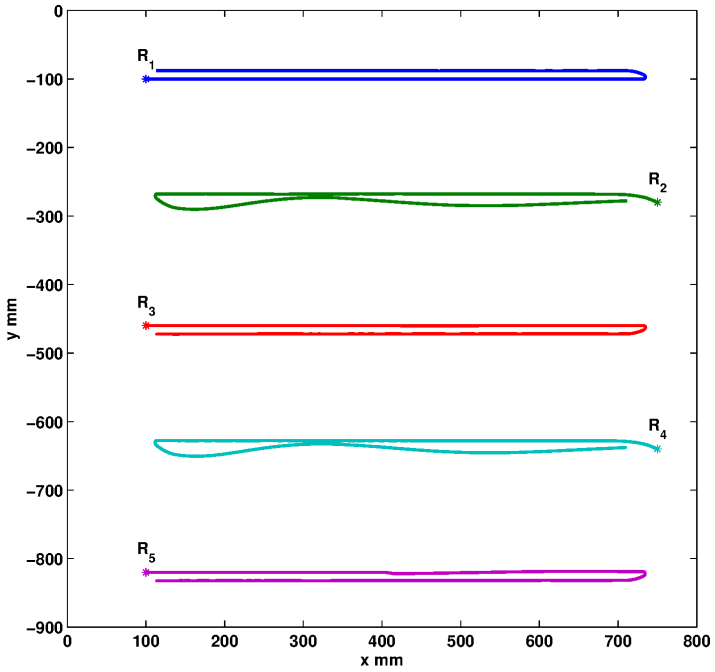
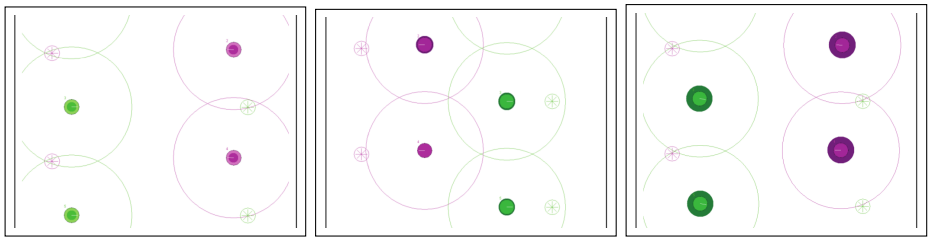


Figura 6.3: Trayectorias lineales cíclicas del grupo G_R con cinco integrantes, prueba de 20min

La figura 6.3 muestra el primer recorrido (ida al punto de destino y vuelta al punto de partida indicado con "*") para los 5 integrantes de G_R. Tres robots inician desde la izquierda del escenario y dos desde la derecha. Cada R_i avanza con una velocidad lineal constante y similar entre ellos por lo que se detectan más o menos de forma regular, a frecuencias similares y aproximadamente a la mitad de su recorrido. Esto permite por una parte, observar la evolución de la covarianza del error cuando el método cooperativo dispone de la información relativa de forma regular y, por otra parte, aporta cierta aleatoriedad al orden de detección de los vecinos entre sí, con lo que se realiza el método cooperativo utilizando tanto información de alta precisión (con el vecino R_1 o con un vecino actualizado recientemente a partir de éste), como de baja (vecinos restantes sin actualización reciente).

Esto resulta relevante en el esquema cooperativo *ECLA* cuando, por ejemplo, se ha alcanzado el límite del evento $R_{A,lim}$ en un R_i, pero éste no dispone de vecinos dentro de D_r ($N_D = 0$) para realizar la actualización con información relativa. En esta situación, el primer vecino que entra en D_r es detectado por R_i, el cual obtiene la medición relativa M_r y la utiliza junto con la información i_R (recibida

del vecino) para corregir su postura. De esta forma, si el primer vecino detectado es R_1 la covarianza del error de estimación de la postura $P_{k,p}$ resultante será baja, lo que sucede también en caso de detectar otro R_i con covarianza del error baja en su postura. Por otra parte si el vecino detectado tiene error alto en la estimación de su postura, se producirá una disminución en la $P_{k,p}$ de R_i, pero esta reducción será de menor magnitud, por lo que $R_{A,lim}$ será alcanzado antes que en el caso de detección con R_1. Este tipo de variabilidad en la detección es conveniente en la simulación ya que permite observar los distintos casos de detección ocurrentes, ya que no existe una secuencia predefinida.



(a) Baja $P_{k,p}$ (b) Media $P_{k,p}$ (c) Alta $P_{k,p}$

Figura 6.4: Trayectorias lineales cíclicas del grupo G_R con cinco integrantes, prueba de 20min

La interfaz de la simulación se muestra en las figuras 6.4(a), 6.4(b), 6.4(c), donde cada R_i se representa como un pequeño círculo relleno de un color sólido claro con una línea blanca que indica su orientación (A_{R_i}, área del robot) y un número con su respectivo índice i. El rango de detección D_r se representa como una circunferencia centrada en cada R_i. La covarianza del error de posicionamiento $P_{k,p}$ también se representa como un círculo relleno de color sólido pero más oscuro, cuya área es igual a A_{ellip}. La evolución de A_{ellip} se va actualizando y visualizando en tiempo real a medida que se ejecuta la simulación.

Por ejemplo, en la figura 6.4(a), cuando la ejecución acaba de empezar, A_{ellip} tiene un valor muy bajo, tal y como muestran los pequeños círculos oscuros. A medida que los robots se mueven, la covarianza aumenta y con ello A_{ellip} se visualiza con círculos mayores (figura 6.4(b)). En el caso del ECLA, cuando se supera el umbral del evento ($A_ellip > A_{R_i}$),figura 6.4(c)), los círculos se vuelven de un color más oscuro, advirtiendo de que se debe producir la actualización cooperativa cuando otro R_i se encuentre dentro del alcance de D_r.

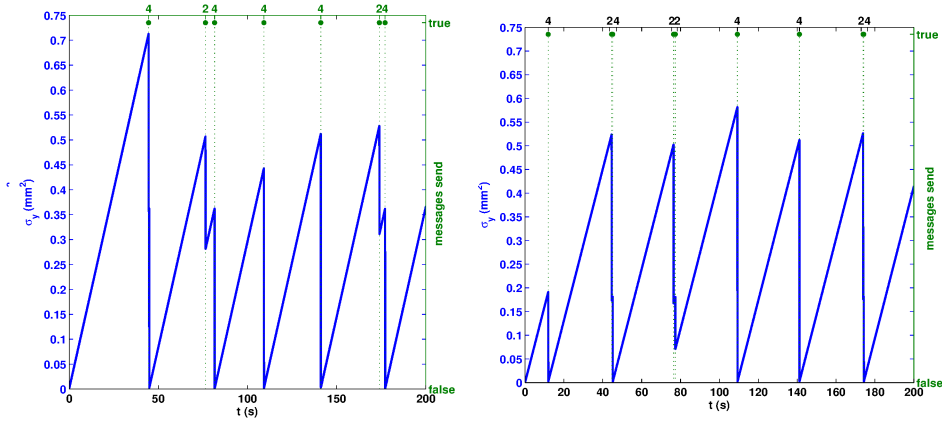
Para analizar el comportamiento y la ejecución de los dos esquemas de localización (TCLA y ECLA), se monitorizó la evolución de la covarianza $P_{k,p}$ para cada robot R_i en una prueba de 20 minutos de duración. Para el caso del ECLA se trabajó con los niveles de evento $R_{A,lim} = \{11.2, 5.6, 3.73\}$, tal y como muestran las figuras 6.5(a), 6.5(b) y 6.6(a), donde se muestra la evolución de $P_{k,y}$ para el robot R_2. La figura 6.6(b) refleja los resultados ejecutando el algoritmo TCLA. En cada gráfica se muestra en el eje superior la cantidad de mensajes intercambiados entre agentes para llevar a cabo la actualización de la localización cooperativa.

Observando con detenimiento las tres figuras sobre el algoritmo ECLA, se puede apreciar cuándo la actualización de posición proviene de R_1 ya que $P_{k,p}$ disminuye casi a cero dado que R_1 es el único agente definido con acceso a información global y por tanto con un error de posición nulo. También se observa el caso alternativo donde la actualización se realiza mediante R_3 y donde $P_{k,p}$ a penas disminuye su valor. En estos casos, el límite $R_{A,lim}$ se vuelve a superar rápidamente y puede coincidir con que en esa segunda consulta, R_1 también se encuentre dentro de D_r por lo que a la hora de elegir entre los dos, siempre escoge la información con menor valor de covarianza en su postura, es decir, la de R_1.

En las dos gráficas de la figura 6.6 se observa que tanto en el ECLA como en el TCLA, el error de estimación está acotado ya que la covarianza no crece indefinidamente debido al uso del método cooperativo y a la utilización de sensores con la precisión adecuada. Además, la evolución de $P_{k,y}$ es similar para ambos métodos ya que las condiciones de simulación son muy semejantes.

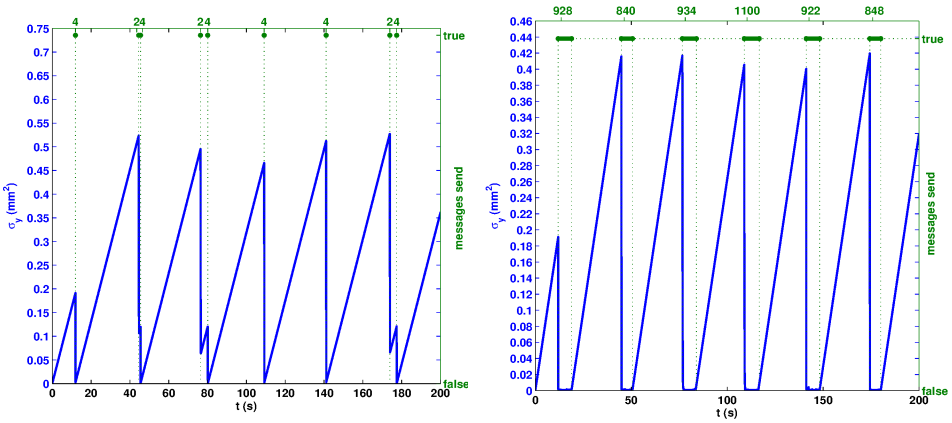
La principal diferencia que se observa entre las dos estrategias radica en la cantidad de mensajes intercambiados entre agentes cuando se requiere realizar una actualización. El método ECLA requiere entre 2 y 4 mensajes por actualización en el peor de los casos (2 cuando sólo se detecta un vecino, 4 cuando se detectan dos a la vez) mientras que el TCLA utiliza del orden de unos 800 o 1000 mensajes (comunicación permanente mientras hay detección entre vecinos) como se observa en las dos gráficas de la figura 6.6.

Esta diferencia del uso del ancho de banda de las comunicaciones es uno de los aportes fundamentales del método de localización cooperativa por eventos. La definición de una *consciencia* del robot para determinar cómo de buena es su localización, permite que él mismo pueda solicitar una actualización, únicamente cuando él considere que la requiere.



(a) ECLA, $R_{A,lim}$ =11.2 (b) ECLA, $R_{A,lim}$ =5.6

Figura 6.5: Evolución de la covarianza de R_2 y mensajes intercambiados, G_R con cinco robots, ECLA con $R_{A,lim} = \{5.6, 11.2\}$, primeros 3min de la prueba de 20min



(a) ECLA, $R_{A,lim} = 3.73$ (b) TCLA

Figura 6.6: Evolución de la covarianza de R_2 y mensajes intercambiados, G_R con cinco robots, ECLA con $R_{A,lim} = 3.73$, $A_{Ri} = 0.9136$ y $TCLA$, primeros 3min de la prueba de 20min

Otra cuestión a evaluar a continuación, es la influencia del valor definido de $R_{A,lim}$. Para ello se obtiene la media de la covarianza del grupo G_R: $\bar{P}_{k,y}$, la cual se calcula al obtener el promedio, en cada instante k de muestreo, de las cinco $P_{k,y}$ de G_R (5 integrantes, una $P_{k,p}$ por cada R_i), tal y como se muestra en la figura 6.7. Además, es conveniente observar la evolución temporal de la media $\bar{P}_{k,y}$, para lo cual se calcula la media acumulativa en el tiempo: $c\bar{P}_{k,y}$ (covarianza promedio acumulativa), al obtener el promedio utilizando todos los valores disponibles de $\bar{P}_{k,y}$ desde el instante inicial de la prueba hasta el instante actual k, tal y como se muestra en la figura 6.8. Por ejemplo, el valor de $c\bar{P}_{k,y}$ en $k = 1s$ es la media de todos los valores de $\bar{P}_{k,y}$ en el intervalo $k = [0, 1s]$.

A partir de las figuras 6.7 y 6.8 se puede observar con claridad la influencia de $R_{A,lim}$ en la precisión del método cooperativo basado en eventos. Al disminuir el valor de $R_{A,lim}$ en *ECLA*, los límites de $P_{k,y}$ también decrecen, tendiendo al valor de los límites de la covarianza obtenidos con el método *TCLA* (se incrementa la precisión del método conforme decrecen los valores máximos de $P_{k,y}$). Sin embargo, este incremento en la precisión tiene el coste asociado de incrementar el número de mensajes utilizados para realizar la localización cooperativa (requiere, por tanto, más recursos computacionales). Con este resultado, se puede realizar el ajuste del valor de $R_{A,lim}$ de acuerdo a la cantidad de recursos disponibles en la plataforma y según la precisión deseada en la estimación de la postura.

Por otra parte, la evolución temporal de $\bar{P}_{k,y}$ representada mediante $c\bar{P}_{k,y}$ (figura 6.8) muestra el comportamiento estable de la estimación, en cuanto el error de estimación es acotado y su covarianza promedio (de G_R) tiende a un valor constante (no crece indefinidamente). En el caso de la localización cooperativa, se alcanza este valor constante debido a la presencia de una fuente de información global con precisión adecuada (en el caso de esta prueba es el robot R_1), y al intercambio de información de los algoritmos *TCLA* y *ECLA* que distribuyen la información global a todos los robots del grupo de forma directa (si se encuentran dentro del D_r) o de forma indirecta (mediante los robots recientemente actualizados por el algoritmo cooperativo).

Finalmente, se observa de la figura 6.8 el comportamiento similar de las estrategias *TCLA* y *ECLA*, siendo ambas acotadas en cuanto al error y covarianza del error de estimación, pero en el caso de *ECLA* se puede realizar el ajuste de la precisión del método y del ancho de banda utilizado para obtener la estimación de la postura.

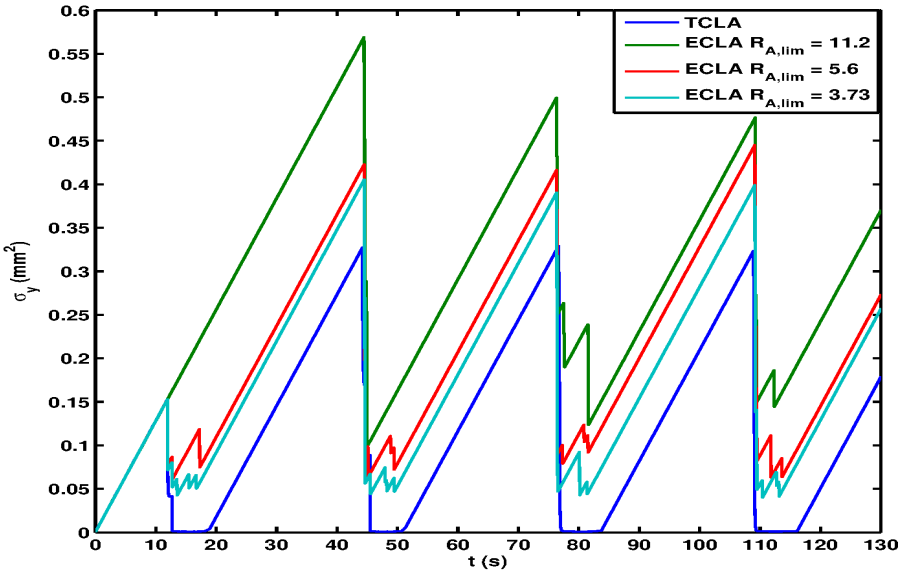


Figura 6.7: Covarianza Promedio $\bar{P}_{k,y}$ para $G_R = 5$, prueba de 20min

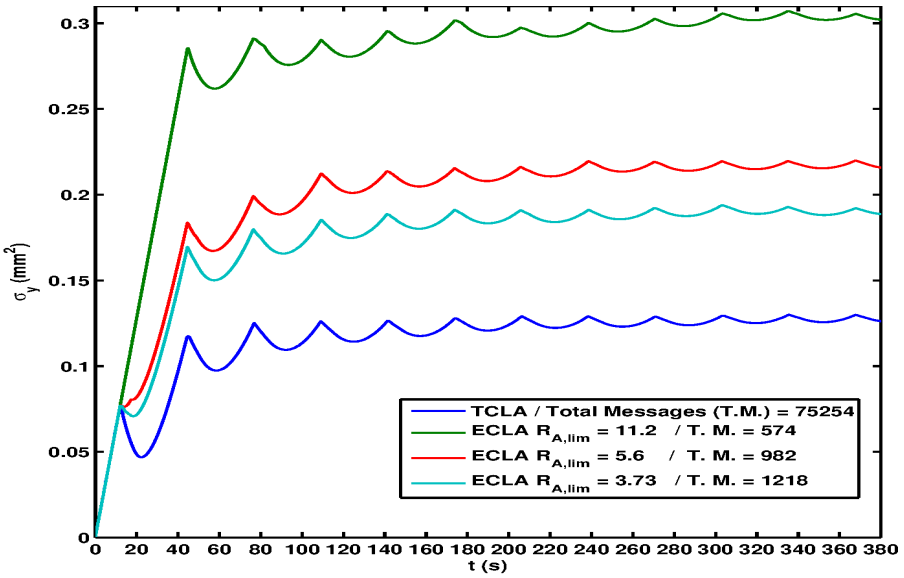


Figura 6.8: Covarianza promedio acumulativa $c\bar{P}_{k,y}$ para el G_R con 5 robots, prueba de 20min

A continuación, la tabla 6.1 muestra el resumen global de la prueba de 20 min. con 5 robots utilizando TCLA y ECLA con los distintos valores de evento($R_{A,lim} = \{3.73, 5.6, 11.2\}$). Se observa la cantidad de mensajes intercambiados y los valores promedio $\bar{P}_{k,x}$ y $\bar{A}_{ellip}$ para la duración total de la prueba de cada R_i dentro del grupo (promedio temporal). Se expone además el promedio del grupo G_R (sin considerar los valores de R_1) y el número total de mensajes intercambiados en G_R.

Tabla 6.1: Resumen de resultados, prueba con el G_R de cinco integrantes, TCLA y ECLA con $R_{A,lim} = \{3.73, 5.6, 11.2\}$

R_i	Algoritmo *TCLA*			Algoritmo *ECLA*, $R_{A,lim} = 3.73$		
	$\bar{P}_{k,x}$	$\bar{A}_{ellip}$	Mensajes Intercambiados	$\bar{P}_{k,x}$	$\bar{A}_{ellip}$	Mensajes Intercambiados
1	0	0	18848	0	0	76
2	0.1622	4.5818	9546	0.2398	6.7536	202
3	0.1589	4.4883	18768	0.2419	6.8102	370
4	0.1589	4.4911	18762	0.2483	6.9951	346
5	0.1705	4.8139	9330	0.2457	6.9190	224
G_R	0.1627	4.5938	75254	0.2439	6.8695	1218

R_i	Algoritmo *ECLA*, $R_{A,lim} = 5.6$			Algoritmo *ECLA*, $R_{A,lim} = 11.2$		
	$\bar{P}_{k,x}$	$\bar{A}_{ellip}$	Mensajes Intercambiados	$\bar{P}_{k,x}$	$\bar{A}_{ellip}$	Mensajes Intercambiados
1	0	0	68	0	0	50
2	0.2534	7.1396	166	0.3106	8.7603	110
3	0.2703	7.6138	292	0.4346	12.2663	118
4	0.2911	8.2092	286	0.4001	11.2925	202
5	0.2920	8.2343	170	0.4634	13.0857	94
G_R	0.2767	7.7992	982	0.4022	11.3512	574

Se puede apreciar la evolución del promedio $\bar{P}_{k,x}$ y $\bar{A}_{ellip}$ los cuales tienden a decrecer conforme se disminuye el valor de $R_{A,lim}$ por las razones anteriormente descritas (para el caso de $\bar{P}_{k,y}$ en las figuras 6.7 y 6.8). Se puede observar nuevamente el compromiso entre la precisión del método y el ancho de banda (coste) requerido para obtenerla, ajustable en *ECLA* según la selección de $R_{A,lim}$.

En robots de recursos limitados, el $R_{A,lim}$ puede ser modificado según el ancho de banda disponible en la plataforma, para realizar un menor número de interacciones con los vecinos dentro de D_r (y por tanto un menor número de ejecuciones de la corrección del filtro de fusión, reduciendo el uso de recursos), pero esto tiene el coste asociado del incremento en la covarianza del error de estimación (y del error en la postura).

6.3.2.2 Grupo de Diez Robots Braitenberg

Dada la importancia de la frecuencia de detección entre agentes vecinos, se considera un nuevo escenario donde, al contrario que en el anterior, un grupo de diez robots se mueven libremente y de manera aleatoria ejecutando un algoritmo Braitenberg de evasión de obstáculos para evitar las posibles colisiones con otros robots o con las paredes que cercan el escenario. Esto genera que las detecciones entre vecinos se produzcan sin un orden predeterminado y que puedan ocurrir casos en los que un robot quiera solicitar una actualización pero no encuentre un vecino con quien hacerlo durante cierto tiempo, o lo contrario, que en lugar de detectar dos vecinos en el momento de la actualización se detecten más de dos a la vez.

En este caso se ha considerado que existen dos robots con acceso a mediciones globales de posicionamiento: R_1 y R_6 ($P_{k,p} = 0$) para proveer la convergencia global al grupo según se especifica en [150] (evitando el crecimiento no acotado del error de estimación de la *pose*). A medida que se lleva a cabo la ejecución, los robots avanzan, se detectan, evitan colisionar ante cualquier obstáculo y ejecutan el algoritmo de localización cooperativa ECLA, actualizando su posición con sus vecinos cuando ellos mismos lo consideran oportuno ($R_A > R_{A,lim}$).

La figura 6.9(b) muestra el movimiento de los robots durante el primer minuto de simulación. La gráfica 6.9(a) muestra la evolución de la covarianza $P_{k,p}$ del robot R_2 durante los 18 primeros minutos de ejecución y las pequeñas gráficas (6.9(c), 6.9(d), 6.9(e), 6.9(f), 6.9(g), 6.9(h), 6.9(i)) muestran la del resto de robots sin acceso al sensor global de posición.

Nuevamente, se observa el buen desempeño del algoritmo distribuido y cooperativo por eventos (ECLA) ya que el error de estimación está acotado y no crece indefinidamente. Según N_D y i_R, la actualización de la postura produce un mayor decremento en $P_{k,p}$ si dentro de D_r se incluye un miembro de G_{RA} (R_1 o R_6 en esta prueba), o si un R_i ha sido recientemente actualizado con la información de los integrantes de G_{RA}, tal y como se observa en detalle para la covarianza $P_{k,y}$ del robot R_2 en la figura 6.9(a). Los restantes R_i de recursos limitados (en G_{RL}, sin acceso a la medición global) muestran una evolución similar. Se observan algunos valores altos en $P_{k,p}$ los cuales ocurren cuando se alcanza $R_{A,lim}$ pero $N_D = 0$, debido a que $P_{k,p}$ crecerá hasta que se realice la actualización de posición (cuando se detecte un vecino, luego $N_D > 0$ y se puedan obtener M_r y recibir i_R). Los valores máximos en $P_{k,p}$ dependerán de la configuración del grupo G_R (misión, movimientos a realizar, etc.), del número de $R_i \in G_{RA}$ con acceso a la información global, así como de la frecuencia con la que éstos se encuentran dentro del rango de detección D_r de los $R_i \in G_{RL}$, siendo aspectos que deben ser tomados en consideración durante la implementación y diseño del grupo multi-agente distribuido y cooperativo.

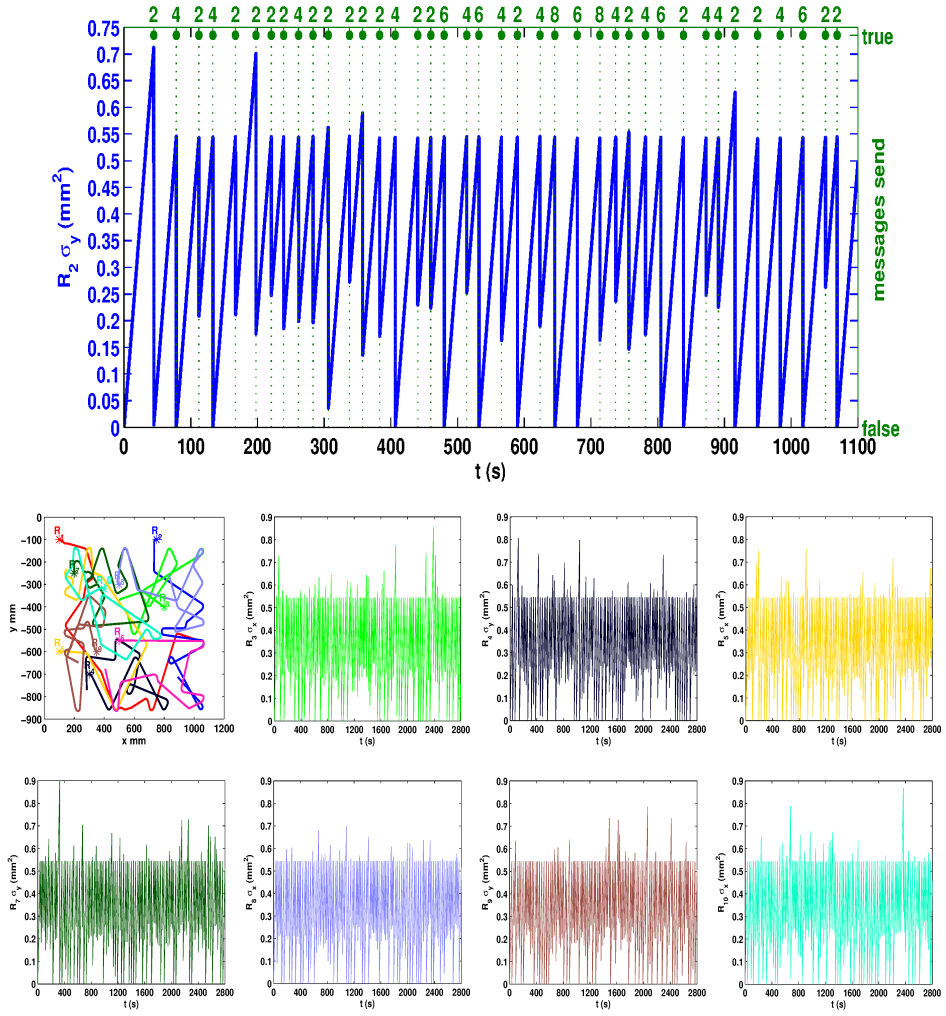


Figura 6.9: *ECLA* con $R_{A,lim} = 16.8$, G_R con $N = 10$, simulación de 46min: movimiento de los robots con la evitación de obstáculos Braitenberg (primer minuto), evolución de la covarianza $P_{k,p}$ y mensajes intercambiados en R_2 (primeros 18min)

6.3.2.3 Análisis de la Elección del Comprometido Valor de $R_{A,lim}$ en el Algoritmo ECLA

Tras las pruebas anteriores queda clara la importancia del valor de $R_{A,lim}$ y su relación directa con el valor promedio mínimo de la covarianza. Menor $R_{A,lim}$ requiere un mayor número de actualizaciones, lo cual provoca un mayor uso del ancho de banda pero a su vez mejora la precisión de la postura (disminuye la covarianza).

Para ver el efecto, se repitió la misma prueba anterior con diez robots utilizando distintos valores de $R_{A,lim}$, obteniendo para cada prueba la covarianza promedio del grupo $\bar{P}_{k,y}$ (promedio para los integrantes de G_R, removiendo R_1 y R_6), y la covarianza promedio acumulativa $c\bar{P}_{k,y}$ (evolución temporal del promedio $\bar{P}_{k,y}$) tal y como se muestra en las figuras 6.10 y 6.11 respectivamente. Como es de esperarse, el esquema de localización cooperativo *ECLA* muestra un comportamiento estable para todos los valores de $R_{A,lim}$ utilizados, en cuanto el error de estimación es acotado y la covarianza del error de estimación no crece indefinidamente (su promedio alcanza un valor estable según la figura 6.11).

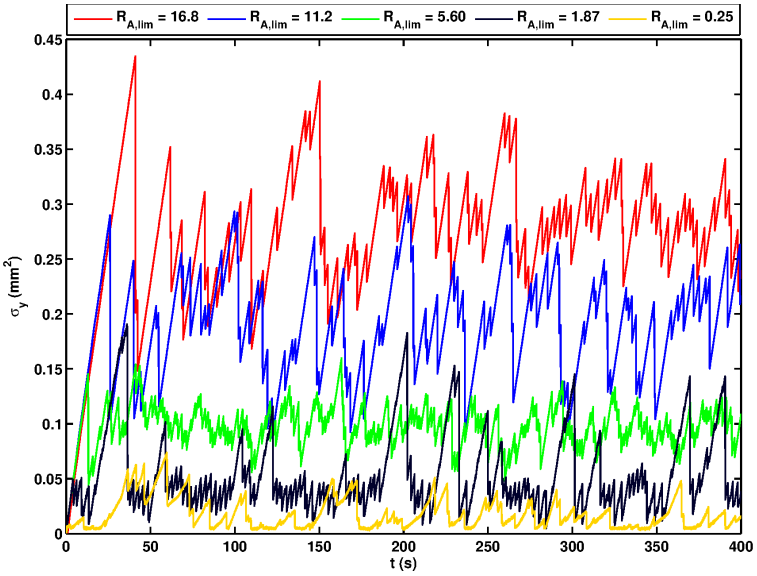


Figura 6.10: Covarianza Promedio $\bar{P}_{k,y}$, evolución para el G_R con diez robots, prueba de 46min

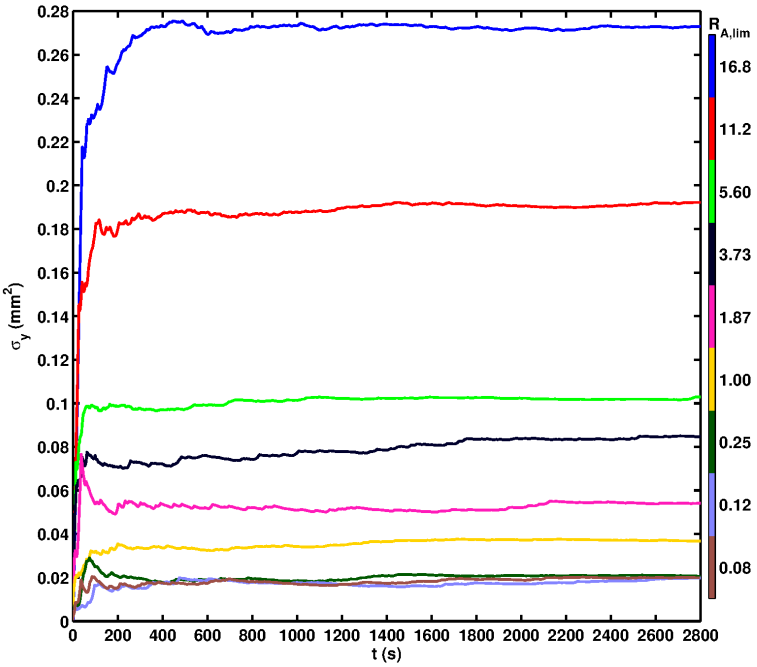


Figura 6.11: Covarianza promedio acumulativa $c\bar{P}_{k,y}$, para el G_R con diez robots, prueba de 46min

Se observa además que se obtiene una mejor precisión del método (mayor uso del ancho de banda) cuando se utilizan valores de $R_{A,lim}$ bajos, en el caso contrario (incremento en $R_{A,lim}$) se disminuye la precisión pero se realizaría un menor intercambio de mensajes entre integrantes del grupo. Con esto se obtiene un resultado similar a las pruebas para el grupo de cinco integrantes, figuras 6.7 y 6.8.

El compromiso entre la precisión del método (promedio de $P_{k,y}$ de todos los robots para la duración total de la prueba de 46min: $T\bar{P}_{k,y}$) y el uso del ancho de banda de comunicación (total de mensajes intercambiados por los robots en G_R en la duración total: *T.M.*) ante la variación del límite del evento $R_{A,lim}$, realizado para la prueba con la evitación de obstáculos Braitenberg se muestra en la figura 6.12 (similar a la del robot diferencial, figura 5.4).

Esta figura muestra la influencia de $R_{A,lim}$ en la precisión y en el número de mensajes intercambios que se requieren para realizar la actualización pertinente. Como ya se preveía, al disminuir $R_{A,lim}$ también disminuyen los márgenes de error (covarianza $T\bar{P}_{k,y}$) del método *ECLA* hasta un límite, definido por el máximo número de consultas que la plataforma puede realizar (según los recursos disponibles). Por el contrario, cuando se incrementa $R_{A,lim}$, se disminuye la precisión del método (se incrementa $T\bar{P}_{k,y}$) pero con la ventaja de reducir la cantidad de mensajes

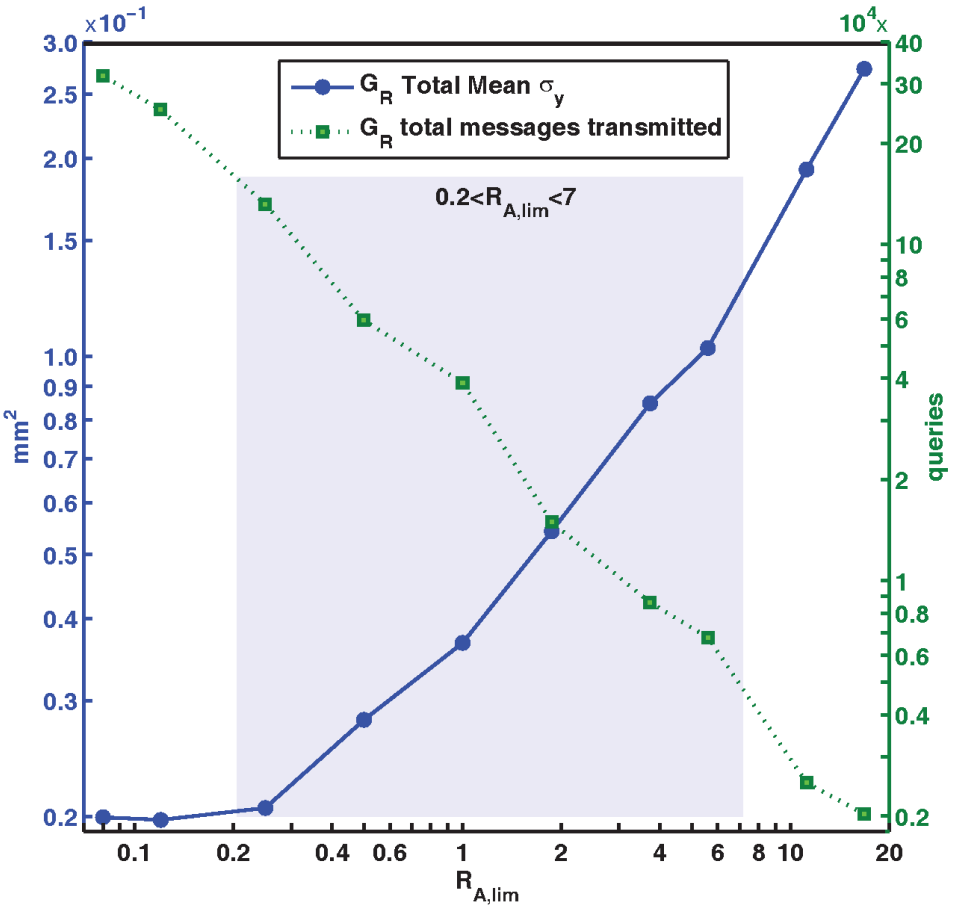


Figura 6.12: Compromiso entre el desempeño (covarianza total $T\bar{P}_{k,y}$) y el ancho de banda (total de mensajes T.M.) definido por el límite del evento $R_{A,lim}$ para el G_R con diez robots, prueba de 46min

intercambiados (*T.M.*) y por lo tanto los recursos computacionales requeridos para la localización distribuida del grupo de robots.

A partir de la dinámica de la figura 6.12, se puede escoger un rango adecuado para $R_{A,lim}$, de forma que se obtenga una solución de compromiso entre la precisión requerida en la localización cooperativa del grupo de robots y la cantidad de recursos computacionales y ancho de banda disponibles en las plataformas (en cada R_i del grupo).

Además, el valor utilizado para $R_{A,lim}$ puede ser definido a voluntad durante la etapa de diseño del sistema para predefinir el máximo intercambio de mensajes deseado, o bien podría modificarse dinámicamente durante la ejecución del programa en cada robot. Por ejemplo, se puede ajustar el valor de $R_{A,lim}$ según los recursos (en tiempo de cómputo y ancho de banda) disponibles en tiempo real en la plataforma, disminuyendo su valor en caso de que todo el ancho de banda de la plataforma esté disponible, o bien incrementar $R_{A,lim}$ si se requieren recursos computacionales o ancho de banda para otras tareas distintas a la localización cooperativa.

Por ejemplo, el rango $0.2 \leq R_{A,lim} \leq 7$ sombreado en la figura 6.12 produce una variación de $0.0204 \leq \bar{TP}_{k,y} \leq 0.1262$ en la precisión y $158265 \geq T.M. \geq 4904$ en el ancho de banda. De esta forma, el rango de $R_{A,lim}$ representa una reducción en la $\bar{TP}_{k,y}$ con $R_{A,lim} = 0.2$ cercana al $16.2\,\%$ del valor de $\bar{TP}_{k,y}$ con $R_{A,lim} = 7$, entretanto el esquema cooperativo con $R_{A,lim} = 7$ utiliza únicamente un $3.1\,\%$ de los mensajes requeridos cuando $R_{A,lim} = 0.2$.

6.3.3 Robots Summits XL con trayectorias lineales cíclicas

Visto el buen desempeño del algoritmo en simulación y para concluir este capítulo de la tesis, se llevó a cabo la implementación del algoritmo ECLA en dos plataformas de robots diferenciales Summit XL.

Además del sistema láser (Hokuyo) embarcado para medir distancias, se instaló una cámara RGB en cada robot y se adjuntaron códigos QR identificativos en sus laterales para otorgarles la capacidad de detectarse e identificarse tal y como muestra la figura 6.13.

Partiendo de la implementación del KF por eventos en el Summit XL, se añadió un nuevo nodo de ejecución basado en el paquete de *ROS vision_vist* para reconocer los códigos QR capturados por la cámara embarcada y posicionarlos e identificarlos. Este nodo publica por el tópic *pose_detected* los resultados de la detección y es a este topic al que se suscribe el nodo de control. Cuando la covarianza supera el umbral $R_{A,lim}$ definido, utiliza los valores leídos del tópic *pose_detected* y los valores del láser para conocer su medida relativa. Las comunicaciones entre los robots se realizan de nuevo con el middleware ROS-JADE mediante el intercambio de mensajes ACL.

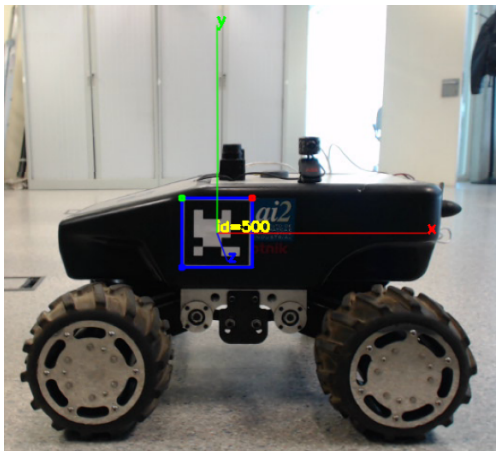


Figura 6.13: Pegatinas QR identificativas instaladas en el Summit XL para facilitar la detección relativa

Además, R_1 tiene acceso al sensor global (en este caso una cámara cenital la cual detecta y posiciona un señuelo triangular situado encima del robot). Del mismo modo que en la primera prueba de simulación, los robots siguen una trayectoria lineal cíclica (mostrada en la figura 6.14) detectándose entre sí cada cierto intervalo de tiempo y realizando la actualización de posición en caso de ser necesario.

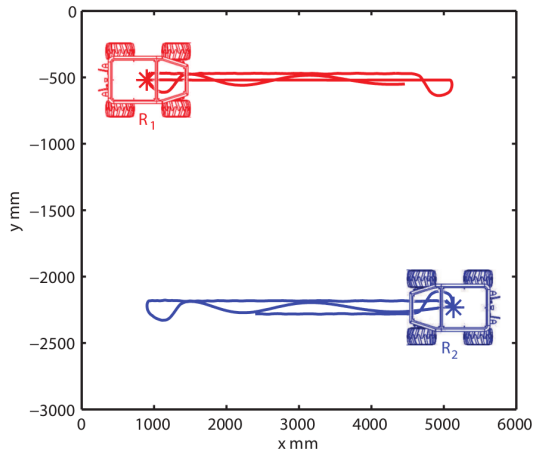


Figura 6.14: Trayectoria lineal cíclica seguida por los dos Summits XL

Para esta prueba no se consideró ningún entorno dinámico (sólo los robots eran detectados por el láser). Cabe destacar la complejidad de conseguir experimentalmente aislar todos los problemas e imprevistos que surgen durante las pruebas empíricas con robots reales para poder evaluar objetivamente el desempeño del algoritmo de localización distribuida y cooperativa, ECLA.

A partir de esta prueba, se obtuvo el área de las elipsoides del error de estimación para el robot sin acceso a la información global, R_2, cuya evolución se muestra en la figura 6.15. Los resultados mostrados en la figura confirman el buen desempeño del ECLA ya que el incremento del error de localización está acotado como se esperaba.

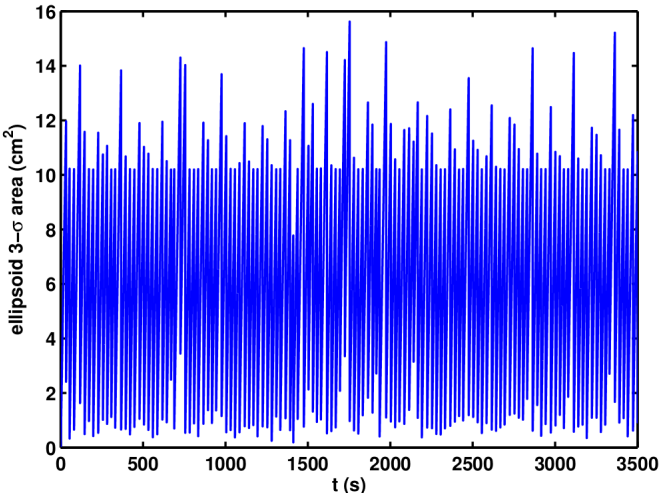


Figura 6.15: Evolución del área de las elipsoides, robot R_2

El *ECLA* propuesto es suficientemente flexible como para ser utilizado en una gran variedad de plataformas. Si existen recursos disponibles, el algoritmo puede adaptarse para utilizar un filtro de fusión sensorial más complejo como el UKF sin reducción para mejorar la precisión de la posición estimada al utilizar modelos altamente no lineales.

Finalmente, en la dirección `/https://www.youtube.com/watch?v=d9uV3iAmI14` se puede encontrar un vídeo resumen de las pruebas realizadas tanto en simulación como con los dos summits de la figura 6.16.

Concluidas todas las pruebas experimentales de los diversos algoritmos propuestos y realizado el análisis de los resultados obtenidos, se procede a exponer las principales conclusiones del presente capítulo, según se describe en la siguiente sección.

Figura 6.16: Robots Summit XL empleados en las pruebas experimentales

6.4 Conclusiones del capítulo

En este capítulo se han expuesto las aportaciones y contribuciones de la presente tesis a la mejora de la localización local y global de robots móviles con recursos limitados mediante el uso de sistemas multi-agente:

- Frente a las opciones disponibles en la literatura, se ha definido el marco de trabajo del algoritmo propuesto en el presente trabajo de manera detallada definiendo el modelo de holarquía utilizado, la red de comunicaciones y el modelo de medición relativa.
- Se ha presentado un algoritmo eficiente de localización distribuida basada en eventos y con procesos cooperativos entre agentes robóticos. Adicionalmente a las posibles misiones definidas entre los agentes, se ha dotado al sistema de la capacidad de cooperar para conseguir mejorar la localización del conjunto de robots a partir de mediciones de las distancias relativas entre agentes.
- A dicho algoritmo se le ha aplicado un método de actualización continua y un método de actualización por eventos, resultando dos algoritmos (TCLA y ECLA, respectivamente) de comportamientos dispares en cuanto a consumos del canal de comunicación y exactitud en la localización.
- Se ha realizado la comparativa entre el método de localización cooperativa temporal (TCLA) y el método por eventos (ECLA), demostrando empíricamente las ventajas del ECLA en cuanto a la reducción del uso del canal de comunicación y la mejora de la precisión de la localización, consiguiendo una solución de compromiso entre estos dos factores.
- Finalmente, se han presentado pruebas experimentales de la implementación de los métodos de localización sobre distintos tipos de robots con resultados satisfactorios que demuestran y validan las aportaciones de los algoritmos desarrollados en el presente capítulo.

Capítulo 7

Navegación y Coordinación de Robots mediante Sistemas Distribuidos

Durante los capítulos anteriores se ha abordado ampliamente el problema de la localización de robots tanto a nivel individual como de un colectivo. La siguiente problemática sobre la que se plantea dar solución mediante estrategias basadas en sistemas distribuidos es la navegación y coordinación de grupos de robots móviles.

En este capítulo se describe una estrategia de navegación mediante planificadores locales y globales con detección y evasión de colisiones desarrollada mediante una arquitectura distribuida y una estrategia de coordinación basada en MAS y aplicada de nuevo a mejorar la localización y, por tanto, la navegación de robots móviles de recursos limitados.

Se presenta el capítulo por orden cronológico a su desarrollo, en primer lugar se describe el algoritmo de planificación de movimientos con evasión de colisiones de objetos estáticos aplicados a robótica móvil y en segundo lugar se continúa con la coordinación distribuida mediante algoritmos reactivos para llevar a cabo una navegación basada en misiones.

7.1 Algoritmo de Navegación Autónoma Basado en una Arquitectura Distribuida

Se entiende por navegación autónoma a la capacidad de un robot para desplazarse a través de un escenario por sí solo. Históricamente la navegación es uno de los campos que más esfuerzos ha concentrado dentro de la comunidad investigadora en robótica y hoy en día es un área muy madurada. Para el caso concreto de la robótica móvil, la problemática del espacio de trabajo se reduce al plano 2D por inherencia del sistema, puesto que el robot sólo es capaz de moverse en el plano XY. La precisión de un algoritmo de navegación autónoma es directamente proporcional a la tasa de éxito en la resolución de las siguientes tareas:

Localización: Tal y como se ha expuesto en capítulos anteriores, para que el robot se desplace de un lugar a otro es fundamental saber dónde se encuentra en cada momento y la *localización* es el proceso por el cual el robot obtiene su posición respecto a una referencia determinada.

Mapping o percepción del entorno: La metodología de percepción del entorno (conocida normalmente como Mapping) es también un aspecto crítico para la navegación ya que está ligada estrechamente a las tareas de localización y planificación. Normalmente la instrumentación más extendida para la percepción del entorno son sensores láser para el reconocimiento geométrico del entorno y cámaras para la detección de balizas u otros objetos de interés. Existen también multitud de metodologías que abordan esta tarea; en algunos casos se trabaja sobre un mapa global previo conocido y en otros se va construyendo poco a poco según el robot recorre el escenario.

Planificación de trayectorias: Esta tarea también es conocida como *pathplanning* y requiere, en la mayoría de los casos, partir de un mapa conocido o al menos parcialmente conocido y preferiblemente estático. El problema a resolver es encontrar un camino factible desde un punto inicial del mapa a otro punto objetivo, evitando cualquier obstáculo que esté situado en el camino. Existen diversas estrategias para resolver este problema como las búsquedas basadas en mapas discretizados en cuadrículas (Grid-Based Search), búsquedas geométricas, algoritmos potenciales de atracción y repulsión, etc.

Estrategias de seguimiento de trayectorias: Una vez que se obtiene la trayectoria del camino que el robot debe seguir, es necesario realizar el control de su seguimiento, y es esa la problemática que abarca esta tarea. Partiendo de la posición actual donde se encuentra el robot, esta tarea debe calcular las acciones de control necesarias para hacer que el robot alcance la trayectoria planificada y la siga fielmente hasta alcanzar un objetivo o posición final. Es una tarea que depende prácticamente de todas las anteriormente descritas y que para obtener buenos resultados debe tener en cuenta parámetros como la configuración del robot

con el que se trabaja, su cinemática, la fiabilidad de la localización, etc. Entre las estrategias más populares del control de seguimiento de trayectorias se encuentra la persecución pura y el punto descentralizado.

La combinación de las tareas descritas anteriormente puede variar según el tipo de algoritmo de navegación que se desee implementar o según los recursos de instrumentación o computación de los que disponga el robot. Como muestra el gráfico de la Figura 7.1, los algoritmos que combinan estrategias de localización y *pathplanning*, se agrupan con el término de *localización activa*, los algoritmos que utilizan únicamente el mapping o la detección del entorno para resolver tareas de *pathplanning* se denominan *algoritmos de exploración* y los algoritmos que se basan en la detección del entorno o mapping para localizarse se engloban en algoritmos de *SLAM* (Simultaneous Localization And Mapping). Por último también existe una clase de algoritmos denominados *SPLAM* (Simultaneous Planning Localization And Mapping) los cuales intentan combinan las estrategias de las tres áreas para navegar de manera autónoma. Naturalmente, cuantas más tareas se intenta abordar al mismo tiempo, más capacidad de cómputo es necesaria para su ejecución.

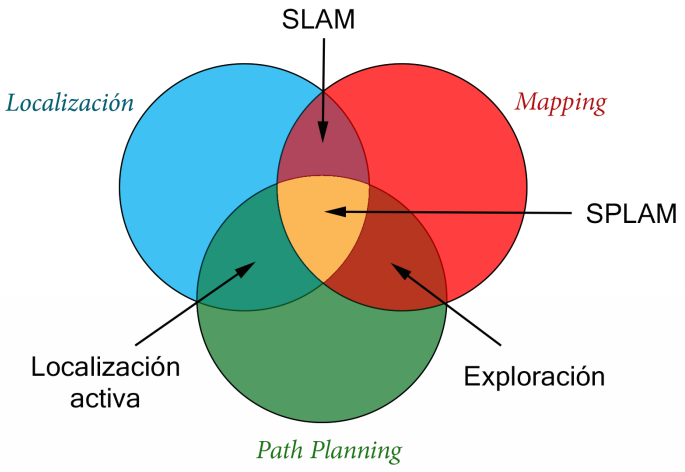


Figura 7.1: Relación entre las distintas tareas de navegación autónoma

El algoritmo que se presenta en esta sección tiene como objetivo ofrecer las tareas de las que depende la navegación autónoma de robots móviles a modo de módulos configurables, reutilizables, escalables e independientes que se comunican entre sí mediante un método de publicación-suscripción, para llevar a cabo una tarea más compleja como es la navegación autónoma. En este caso, se presenta un algoritmo basado en la navegación autónoma sobre un entorno dinámico parcialmente conocido.

Se han definido una serie de agentes software en módulos de ejecución organizados concretamente como nodos del *stack* de navegación de ROS [141]. Como es habitual, cada nodo se ejecuta de forma independiente al resto, no obstante existe una serie de dependencias entre algunos módulos que les obliga a comunicarse entre ellos para poder ofrecer resultados tal y como se expone a continuación.

7.1.1 Metodología del Algoritmo de Navegación Propuesto

El algoritmo de navegación propuesto basa la localización y la planificación global de las trayectorias de los robots en técnicas que parten de un mapa del entorno parcialmente conocido. Por este motivo, el primer algoritmo desarrollado, se centra en la construcción de dicho mapa mediante técnicas de SLAM. Una vez obtenido el mapa parcial del entorno, el siguiente paso es obtener una buena estimación de la localización del robot, por lo que se detalla el algoritmo de localización desarrollado basado en sistemas de partículas (algoritmo de Montecarlo). Por último, resuelta la localización, se describe el algoritmo de navegación y sus planificadores de trayectorias de movimiento.

7.1.1.1 Descripción del Algoritmo de SLAM

En el nivel más bajo del algoritmo de obtención del mapa del entorno, se encuentran los módulos que gestionan los dispositivos hardware disponibles en el robot y que no requieren ninguna información de entrada si no que se limitan a publicar las lecturas que ofrecen los dispositivos que gestionan. Para este trabajo es el caso de los nodos de los drivers de las ruedas, el láser Hokuyo y el giróscopo que incorpora el robot. Los drivers de las ruedas publican la lectura de los sensores de efecto hall que tienen las ruedas en 4 topics distintos: $front\_right\_wheel$, $front\_left\_wheel$, $back\_right\_wheel$, $back\_left\_wheel$. Las medidas del láser se publican por un topic llamado $/scan$ y las medidas del giro por $/gyro$.

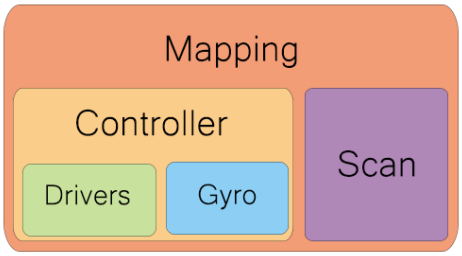


Figura 7.2: Jerarquía de los módulos o nodos que forman el algoritmo de SLAM.

Tal y como muestra la figura 7.2, un nivel por encima de los módulos de *drivers* y *gyro*, se encuentra el módulo de control denominado *Controller*, el cual depende estrictamente de su suscripción al topic del giróscopo para la detección de la velocidad angular del robot y al de los topics de los drivers de las ruedas. Este nodo *Controller* ofrece una primera odometría básica publicada por un topic llamado */Odom*. En este caso no se ha utilizado el módulo de fusión sensorial desarrollado en el capítulo 5 ya que la navegación se realizó en interiores sin opción a un posicionamiento global.

El nodo *Controller* es, además, el encargado de recibir las velocidades de referencia que deben alcanzar las ruedas, implementando para ello un controlador PD. Para ofrecer ese servicio, el nodo se suscribe a un topic llamado */cmd* por el cual deberán publicar aquellos nodos que deseen actuar sobre las ruedas.

Una vez obtenida la odometría, es posible integrar el nodo *Mapping* el cual implementa el algoritmo desarrollado para la obtención del mapa. Dicho nodo se suscribe a los topics */scan* del nodo Hokuyo y */Odom* del *Controller* para publicar tanto el mapa que va construyendo (*/map*) como la posición global calculada dentro del mismo (*/Global_Odom*).

El manejo del robot por el escenario se realiza de forma manual mediante el nodo *Mando* quien gestiona la teleoperación del robot mediante un mando o *joystick* inalámbrico conectado al robot vía Bluetooth. Este nodo publica las velocidades deseadas por el topic */cmd* de forma que son recibidas por el nodo suscriptor *Controller*. De esta manera, la estructura de los módulos que componen este algoritmo de SLAM queda como se muestra gráficamente a continuación:

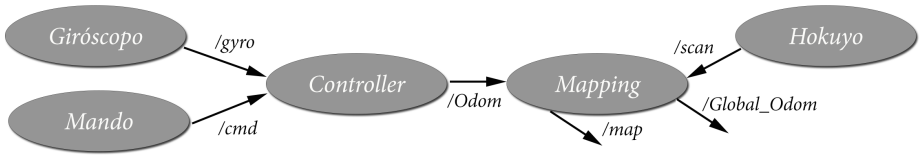


Figura 7.3: Interconexiones entre los módulos o nodos que forman el algoritmo de SLAM.

El algoritmo de mapping implementado se basa en el filtro de partículas de Rao-Blackwellized definido en [127], el cual utiliza la factorización de la ecuación 7.1 para calcular la posición actual del robot en cada momento y la construcción del mapa al mismo tiempo.

Dado el instante t, $m_{1:t}$ representa el mapa construido hasta ese instante, $x_{1:t} = x_1, ..., x_t$ define la trayectoria del robot estimada por el algoritmo, $z_{1:t} = z_1, ..., z_t$ las medidas leídas por el sensor láser y $u_{1:t-1} = u_1, ..., u_{t-1}$ las medidas de la odometría interna calculada por el robot.

$$(x_{1:t}, m_{1:t}|z_{1:t}, u_{1:t-1}) = p(m_{1:t}|x_{1:t}, z_{1:t}) \cdot p(x_{1:t}|z_{1:t}, u_{1:t-1}) \tag{7.1}$$

En la ecuación 7.1 se observa cómo $p(m_{1:t}|x_{1:t}, z_{1:t})$ puede calcularse analíticamente ya que $x_{1:t}$ y $z_{1:t}$ son conocidas, por lo que el grueso del algoritmo recae sobre la estimación de las posibles trayectorias calculadas, $p(x_{1:t}|z_{1:t}, u_{1:t-1})$. Sobre cada partícula del algoritmo se define una trayectoria potencial y además un mapa individual asociado. La clave de los algoritmos de SLAM es la eficiente realización del filtrado de partículas y su correcta ponderación, para finalmente seleccionar la partícula que más se aproxima a la trayectoria real del robot y ofrecerla como resultado.

El proceso para la obtención del mapa se puede resumir en 4 pasos:

1. **Muestreo:** Generación del conjunto de soluciones posibles o partículas, S_{t-1}, compuesto por las tuplas $< x_{t-1}, w_{t-1}, m_{t-1} >$. Donde x_{t-1} define la posible trayectoria estimada para cada partícula, w_{t-1} los pesos asociados a dichas trayectorias y m_{t-1} el mapa generado hasta el instante $t-1$.

2. **Asignación de pesos a las partículas:** A cada partícula i se le asigna un peso según la ecuación 7.2 acorde a una distribución probabilística Gaussiana en el caso que ocupa, aunque podría considerarse otros métodos definidos en la ecuación como ψ.

$$w_t^{(i)} = \frac{p(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})}{\psi(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})} \tag{7.2}$$

3. **Remuestreo:** Se realiza el filtrado de partículas según el peso que tienen asignado. Este paso debe asegurar que el conjunto de partículas siempre es óptimo y finito. Después del remuestreo, se vuelve a asignar a todas las partículas el mismo peso.

4. **Estimación del mapa:** Para cada partícula resultante de la etapa anterior, se calcula su mapa correspondiente $(p(m_{1:t}^{(i)})|x_{1:t}^{(i)}, z_{1:t})$ basándose en la trayectoria $x_{1:t}^{(i)}$ calculada para esa partícula y su historial de observaciones o medidas recogidas $z_{1:t}$.

El algoritmo 14 muestra el pseudocódigo del método implementado para el nodo *Mapping* bajo el sistema ROS.

Algoritmo 14: Algoritmo de SLAM implementado en el nodo *Mapping*

Entrada: $S_{t-1},\ z_t,\ u_{t-1}$
Salida: S_t Inicialización: $S_t = \{\}$

Para $S_{t-1}^{(i)} \in S_{t-1}$ **hacer**
  $< x_{t-1}^{(i)}, w_{t-1}^{(i)}, m_{t-1}^{(i)} >= s_{t-1}^{(i)}$
  //Obtención del nuevo punto $x_t^{(i)}$ de la trayectoria
  $x_t'^{(i)} = x_{t-1}^{i} \oplus u_{t-1}$
  $\hat{x}_t^{(i)} = \underset{x}{\text{argmax}}\ p(x|m_{t-1}^{(i)}, z_t, x_t'^{(i)})$
  Si $x_t'^{(i)} = Fallo$ **entonces**
    $x_t^{(i)} \sim p(x_t|x_{t-1}^{(i)}, u_{t-1})$
    $w_t^{(i)} = w_{t-1}^{(i)} \cdot p(z_t|m_{t-1}^{(i)}, x_t^{(i)})$
  si no
    //Obtención de las medidas del láser cercanas a la posición
    Para $k = 1 \ldots, K$ **hacer**
      $x_k \sim \{x_j || x_j - \hat{x}^{(i)}|\}$
    fin
    //Cálculo de la aproximación Gaussiana
    $\mu_t^{(i)} = (0, 0, 0)^T$
    $\eta^{(i)} = 0$
    Para $x_j \in \{x_1, \cdots, x_K\}$ **hacer**
      $\mu_t^{(i)} = \mu_t^{(i)} + x_j \cdot p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
      $\eta_t^{(i)} = \eta_t^{(i)} + p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
    fin
    $\mu_t^{(i)} = \mu_t^{(i)} / \eta^{(i)}$
    $\kappa_t^{(i)} = 0$
    Para $x_j \in \{x_1, \ldots, x_K\}$ **hacer**
      $\kappa_t^{(i)} = \kappa_t^{(i)} + (x_j - \mu^{(i)})(x_j - \mu^{(i)})^T \cdot p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
    fin
    //Obtención de la nueva postura
    $x_t^{(i)} \sim \psi(\eta_t^{(i)}, \kappa_t^{(i)})$
    //Actualizar la importancia de los pesos
    $w_t^{(i)} = w_{t-1}^{(i)} \cdot \psi^{(i)}$
  fin
  //Actualizar el mapa
  $m_t^{(i)} = integrarScaner(m_{t-1}^{(i)}, x_t^{(i)}, z_t)$
  //Actualizar el conjunto de soluciones
  $S_t = S_t \cup \{< x_t^{(i)}, w_t^{(i)}, m_t^{(i)} >\}$
fin

En cada iteración del algoritmo, se recibe una nueva tupla de valores $< u_{t-1}, z_t >$ (odometría calculada en el instante anterior y medidas del láser para el instante t, respectivamente). Además se cuenta con el conjunto de soluciones o partículas anteriores, S_{t-1}, donde cada partícula se analiza de manera independiente siguiendo los siguientes pasos:

1. Se asume que el nuevo punto x_t^i de la nueva trayectoria calculada con la partícula i se puede obtener a partir de la trayectoria previa calculada y el valor de odometría de entrada, $x_t^i = x_{t-1}^i \oplus u_{t-1}$. Donde el operador $\oplus$ se corresponde con el cálculo de la postura definido en [106].

2. Además, la nueva trayectoria ejecutada sobre el mapa m_{t-1}^i debe coincidir desde el principio con todas las muestras registradas con el láser z_{t-1}. Para esto se utiliza la ecuación 7.3 extraída del algoritmo de correspondencia *vasco* definido en [123]. En caso de que la correspondencia falle, la postura y los pesos de las partículas se asignan teniendo en cuenta únicamente el modelo de movimiento asociado y no las medidas del escáner; saltando la ejecución directamente al paso 5.

$$\hat{x}_t^{(i)} = \underset{x}{\text{argmax}} p(x|m_{t-1}^{(i)}, z_t, x_t'^{(i)} \tag{7.3}$$

3. Se selecciona el conjunto de puntos medidos por el láser cercano al último punto calculado de la trayectoria, x_t^i y se estima el conjunto de posiciones x_j que pertenecen a la frontera del mapa que se desea construir. Además se pondera la importancia de los pesos de dichos puntos mediante el factor de normalización de la ecuación 7.4 donde se obtiene el sumatorio de la probabilidad de acierto o correspondencia de las medidas del láser frente al mapa y las posiciones candidatas respecto a la trayectoria generada hasta el momento.

$$\eta^{(i)} = \sum_{j=1}^{K} p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_j|x_{t-1}^{(i)}, u_{t-1}) \tag{7.4}$$

4. La nueva postura $x_t^{(i)}$ de cada partícula i se incluye dentro de una aproximación Gaussiana definida como $\psi(\mu_t^{(i)}, \kappa_t^{(i)})$

5. Se actualizan la importancia de los pesos de cada partícula. $w_t^{(i)} = w_{t-1}^{(i)}$

6. Y por último se actualiza el mapa $m^{(i)}$ para cada partícula i según la última posición $x_t^{(i)}$ calculada y las medidas del láser z_t.

Mediante este algoritmo se realizó por ejemplo, un mapa del escenario de interiores por donde el robot iba a navegar de manera autónoma posteriormente. En la figura 7.4 se muestra el mapa resultante de una navegación teleoperada del robot Summit XL por el departamento ai2 de la UPV.

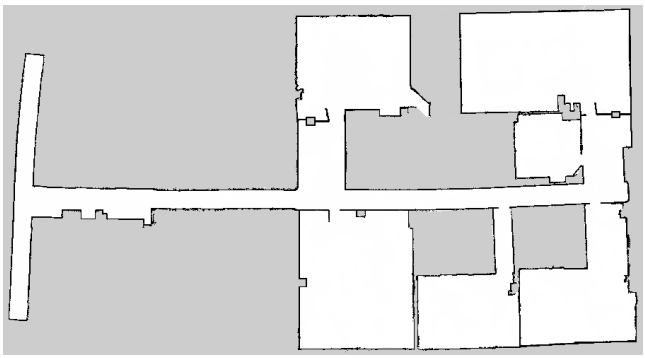


Figura 7.4: Mapa del departamento ai2 realizado con el algoritmo de SLAM.

Alternativamente a este algoritmo, se contó con la oportunidad de poder utilizar un dispositivo de barrido láser 3D modelo $FARO\ FOCUS3D$ como el que se muestra en la figura 7.5 para poder obtener mapas muy precisos del entorno de navegación. Estos dispositivos proyectan haces de luz infrarroja sobre la perpendicular al suelo mientras realizan un giro de 360º sobre su propio eje horizontal, obteniendo así una nube circular de puntos 3D con un diámetro de aproximadamente 7 metros.

Figura 7.5: Escaner 3D FARO Focus 3D para el modelado del entorno del garaje.

Este dispositivo se utilizó para escanear una parte del garaje situado en los sótanos de la Ciudad Politécnica de la Innovación de la UPV. Se realizaron diversos escaneos y finalmente se integraron todos en un único mapa de puntos 3D el cual definía la zona con una precisión de centímetros.

Figura 7.6: Captura del resultado del escáner 3D en el entorno del garaje.

Un vídeo de este experimento puede observarse en la siguiente dirección: `/https://www.youtube.com/watch?v=z1g09Hh8PmY` o capturando el código QR situado a la derecha de este texto.

Debido a que los robots móviles utilizados en la experimentación únicamente llevan incorporado un escáner láser 2D, la nube de puntos obtenida del garaje fue reducida a un mapa 2D cortando a la altura desde el suelo a la que se encontraba instalado el láser en el robot. Además, los coches aparcados fueron eliminados y limpiados del mapa debido a que el objetivo era obtener un mapa estático global del entorno. Los coches no siempre estaban situados en el mismo sitio durante la ejecución de las pruebas por lo que se dejaron reflejados en el mapa únicamente los elementos estructurales de columnas y paredes. La evasión de los obstáculos no recogidos en este mapa es tarea del planificador local tenerlos en cuenta mediante su detección gracias a las medidas recogidas por el láser. Al final de esta sección se detalla dicho procedimiento.

7.1.1.2 Descripción del Algoritmo de Localización del Robot

Una vez conseguido el mapa estático global por donde el robot va a navegar, ya es posible aplicar las técnicas basadas en localización por Montecarlo para situar el robot sobre el mapa conocido, tal y como se expone en [180], [54] y [65].

En términos de software, el cambio de misión se reduce a unos pocos cambios en la estructura de los procesos. El proceso *Mapping* se suspende o se cancela para liberar carga de procesamiento y se lanza el nodo *Map_server* el cual es quien ofrece el mapa creado previamente por el nodo *Mapping* mediante su publicación por un topic llamado */map*. El nodo de localización por Montecarlo recibe el

nombre de *Amcl* y además de la suscripción al mapa global, tiene como entradas la subscripción al topic que ofrece la odometría (*/Odom*)y la suscripción al topic */scan* que ofrece las medidas del láser. El nodo *Amcl* implementa un filtro de partículas basado en Montecarlo ofreciendo como salida un vector con todas las posibles soluciones a la localización global del robot dentro del mapa (topic */particles*) y otra salida con una única posición global del robot por el topic (*/Amcl_Pose*), la cual se corresponde con la posición de la partícula con mejor estimación, es decir, la de mayor peso.

La metodología de localización del *Amcl* es parecida a la del nodo *Mapping*. De hecho, los métodos de SLAM mediante filtro de partículas basan sus algoritmos de localización en los métodos de Montecarlo sólo que además deben cumplir paralelamente la tarea de construir el mapa. Para el caso del *Amcl*, el mapa ya está construido y el objetivo del método se reduce a posicionar al robot dentro del mapa conocido.

La figura 7.7 muestra la interacción de los distintos módulos que participan para conseguir llevar a cabo la localización.

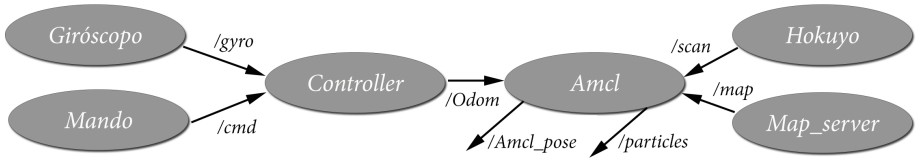


Figura 7.7: Nodos que forman parte del algoritmo de localización por Montecarlo.

El algoritmo 15 detalla el pseudocódigo del *Amcl*. Hay que tener en cuenta que el algoritmo requiere estrictamente un mapa m cuyo perímetro esté completamente definido, sea cerrado, limitado y convexo.

A diferencia del algoritmo 14, en este algoritmo la tupla que compone el conjunto de soluciones factibles se reduce a $< x_{t-1}^{(i)}, w_{t-1}^{(i)} >$. Se pueden distinguir las siguientes seis fases dentro del algoritmo:

1. Del mismo modo que ocurre en el algoritmo de SLAM, al conjunto de partículas existentes de iteraciones anteriores (en caso de haberlas), se les añade un punto nuevo al final de la trayectoria calculado a partir de la trayectoria previa calculada y el valor de odometría de entrada, $x_t^i = x_{t-1}^i \oplus u_{t-1}$. Donde el operador $\oplus$ se corresponde con el cálculo de la postura definido en [106].

2. A este conjunto se le añade un conjunto nuevo generado a partir de una función $\jmath()$ la cual en ocasiones se define mediante algoritmos genéticos, métodos

Algoritmo 15: Algoritmo de localización por MCL implementado en el nodo *Amcl*

Entrada:
m: Mapa generado por Mapping
S_{t-1}*: Conjunto de partículas de la iteración anterior*
z_t*: Medidas del sensor láser*
u_{t-1}*: Medidas tomadas por la odometría interna del robot*
Salida:
S_t*: Conjunto de partículas satisfactorias*
Inicialización: $S_t = \{\}$

Para $S_{t-1}^{(i)} \in S_{t-1}$ **hacer**
  $< x_{t-1}^{(i)}, w_{t-1}^{(i)} >= s_{t-1}^{(i)}$
  //Obtención del nuevo punto $x_t^{(i)}$ de la trayectoria
  $x_t'^{(i)} = x_{t-1}^{i} \oplus u_{t-1}$
  //Adición de posiciones de partículas nuevas
  $x_t = \{x_{t-1} \cup \jmath(x_c)\}$
  Si $x_t'^{(i)} != Fallo$ **entonces**
    //Obtención de las medidas del láser cercanas a la posición
    Para $k = 1 \ldots, K$ **hacer**
      $x_k \sim \{x_j || x_j - \hat{x}^{(i)}|\}$
    fin
    //Cálculo de la probabilidad de correspondencia
    Para $x_j \in \{x_1, \cdots, x_K\}$ **hacer**
      $x_p^{(i)} = p(x_{1:t}^{(i)}, x_j 1 : t^{(i)} | z_{1:t}) \subseteq m$
    fin
    //Actualización de los pesos
    $w_t^{(i)} = w_{t-1}^{(i)} \cdot p(x_{1:t}^{(i)}, x_j 1 : t^{(i)} | z_{1:t})$
    //Filtrado de partículas
    filtro(x_t,$w_t^{(i)}$);
  fin
  //Actualizar el conjunto de soluciones
  $S_t = S_t \cup \{< x_t^{(i)}, w_t^{(i)} >\}$
fin

estadísticos, funciones aleatorias e incluso mediante una combinación de distintos métodos entrelazados, $x_t = \{x_{t-1} \cup \jmath(x_c)\}$.

3. Sobre cada trayectoria considerada $x_{1:t}^{(i)}$, donde el último punto se corresponde con la nueva posición añadida en la fase anterior, se realiza la traslación de las medidas del láser z_t, obteniendo un conjunto de puntos $x_j^{(i)}$.

4. A continuación se define una función de probabilidad para cada conjunto de $x_j^{(i)}$ definida por $p(x_{1:t}^{(i)}, x_j 1 : t^{(i)} | z_{1:t} \subseteq m)$; donde se evalúa la probabilidad de que las medidas relativas hasta el instante t junto con la trayectoria a evaluar, pertenezcan a la medidas del láser hasta el mismo instante; siendo éstas un subconjunto del perímetro del mapa conocido por donde se desplaza el robot.

5. Esta probabilidad se utiliza para ponderar las distintas posibles soluciones calculadas. $w_t^{(i)} = w_{t-1}^{(i)} \cdot p(x_{1:t}^{(i)}, x_j 1 : t^{(i)} | z_{1:t})$

6. Por último, se realiza un filtrado de las partículas para obtener el subconjunto $S_t =< x_t^{(i)}, w_t^{(i)} >$ que mejor se adapta a la posición donde está situado el robot en el instante t.

El algoritmo tiene tres puntos críticos que definen su coste computacional. El más crítico es el número de partículas obviamente, ya que a mayor número, mayor número de iteraciones. Sin embargo también juegan un papel importante las estrategias de ponderación y filtrado de las partículas ya que normalmente implementan algoritmos de correspondencia de alto coste computacional. En la sección 7.1.2 se definen las estrategias utilizadas en los demostradores para poder aplicar este tipo de algoritmos a plataformas de recursos limitados.

7.1.1.3 Descripción del Algoritmo de Navegación Autónoma

Una vez, obtenido el mapa del entorno y una buena estimación de la posición global, el siguiente paso es el desarrollo de un módulo de *pathplanning* que lea la posición local del robot y dada una posición destino calcule la trayectoria libre de obstáculos que debe seguir el robot para alcanzar su objetivo. Para ello, se han implementado dos módulos de planificación independientes: el *Local_Planner* y el *Global_Planner*, según si la navegación es en base a un destino con coordenadas globales dentro de un mapa conocido o un destino relativo a la posición local del robot.

Estos módulos dependen cada uno de su propio mapa de costes donde se especifican los posibles obstáculos que debe tener en cuenta el algoritmo para calcular una trayectoria libre de colisiones hasta el destino. La idea principal es planificar a dos niveles, un nivel global (*Global_Planner*) basado en un mapa de coste estático

(*Global_Costmap*) el cual se genera a partir del mapa conocido del entorno, y un segundo planificador (*Local_Planner*) el cual va actualizando un mapa de coste dinámico (*Local_Costmap*) según la información que recibe de sensores de detección de objetos como el escáner láser. Los mapas de costes están definidos a modo de rejilla o cuadrícula 2D con una resolución menor siempre que el radio del tamaño del robot. A pesar de que el módulo de navegación utiliza el mapa de coste como una estructura bidimensional, en realidad cada celda se representa como un Voxel tridimensional que incluye la descripción del estado de la celda, como su estado (libre, ocupada o desconocida) y su coste. Las celdas ocupadas por obstáculos tienen un coste muy alto, lo cual significa que no se permite que el robot ocupe en ningún caso esa celda durante el recorrido de su trayectoria. Además, alrededor de estas celdas prohibidas se define un perímetro de seguridad (personalizable en cada caso) el cual propaga su coste (reducido exponencialmente) a la celdas vecinas.

La figura 7.8 muestra un ejemplo en el visualizador *rviz* de un mapa de coste global y local donde en amarillo pueden observarse las celdas con mayor coste representando los obstáculos detectados y las celdas en todos morados y azules su propagación a celdas vecinas.

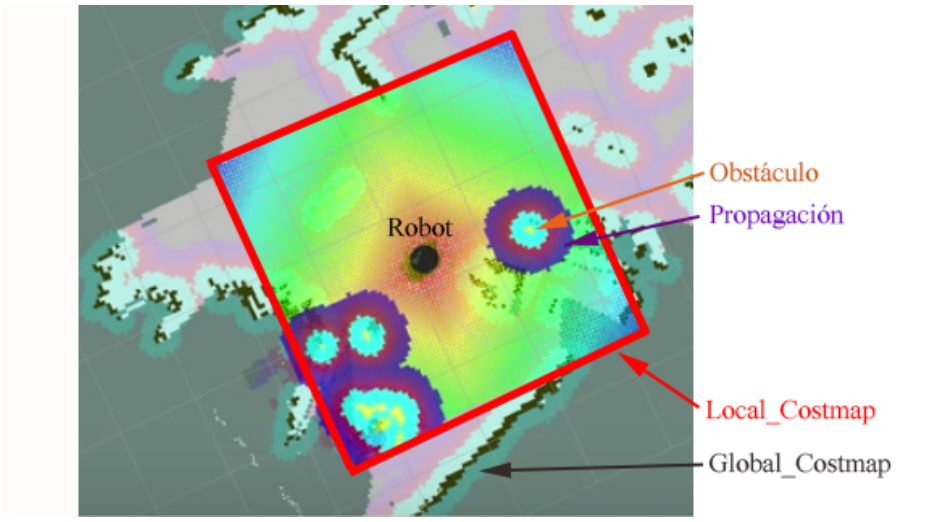


Figura 7.8: Visualización en rviz de los mapas de coste local y global.

Estos mapas de costes los generan los módulos *Local_Costmap* y *Global_Costmap*. Éste último módulo se suscribe al topic */map* ofrecido por el módulo *Map_server* y al topic */scan* ofrecido por el nodo *Hokuyo* para de ese modo publicar un mapa estático de costes global por el topic */costmap_global*. A ese topic se suscribe el nodo *Global_Planner* para calcular una trayectoria global libre de obstáculos

hasta un objetivo determinado dentro del mapa. Dicha trayectoria la publica por el topic */global_path*, al cual está suscrito el nodo *Local_Planner*. Como se ha comentado anteriormente, este nodo tiene también su propio mapa de costes ofrecido por el módulo *Local_Costmap*, que junto a las lecturas del láser y de la odometría (topics */scan* y */Odom*), planifica la trayectoria local necesaria para alcanzar la trayectoria global, teniendo en cuenta los obstáculos dinámicos que puedan ser detectados por el láser. El mismo planificador local es quien finalmente manda las acciones de control necesarias al nodo *Controller* para que el robot siga finalmente la trayectoria calculada. Hay que tener en cuenta que la frecuencia de ejecución del nodo local siempre debe ser mayor que la del nodo global, ya que si no la evasión de obstáculos dinámicos no funcionaría correctamente. También es necesario configurar ciertos parámetros de los planificadores como la distancia que se desea dejar como margen de seguridad respecto a los obstáculos o la fiabilidad con la que debe ceñirse el planificador local al global. La figura 7.9 muestra la arquitectura que describe el algoritmo de navegación autónoma implementado en el robot.

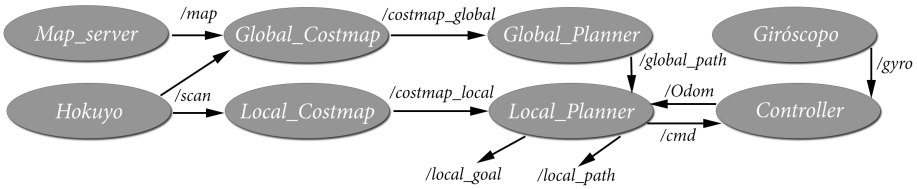


Figura 7.9: Comunicaciones entre los módulos del algoritmo de navegación autónoma.

Planificador Global

El módulo *Global_Planner* recibe la información de los obstáculos y los costes a partir del topic *costmap_global*. La implementación del planificador global se puede resumir en un algoritmo de búsqueda A* el cual planifica una trayectoria desde la celda que ocupa la posición actual del robot en el mapa, hasta la posición de la celda destino, teniendo en cuenta las restricciones definidas por los obstáculos. Este planificador está basado en [91] y no tiene en cuenta la dinámica ni la cinemática del robot. Si bien esto asegura que el planificador sea eficiente y reutilizable para cualquier tipo de plataforma, también lo hace optimista y el camino creado puede no ser factible en todos los casos. Por este motivo se utiliza tan sólo como guía de alto nivel para ofrecer una primera propuesta al módulo de planificación local.

Planificador Local

El planificador local es el responsable de generar los comandos de velocidad que hacen que el robot navegue siguiendo la trayectoria calculada hasta alcanzar su destino. El módulo *Local_Planner* implementa el mismo algoritmo A* para recalcular el planning en caso de que exista colisión e intenta ceñirse a la trayectoria generada por el planificador global teniendo en cuenta la cinemática y la dinámica del robot, así como la información del mapa de costes local. Con el fin de generar unos comandos seguros de velocidad, el planificador utiliza la técnica conocida como *Enfoque Dinámico de Ventana* (DWA del inglés: *Dynamic Window Approach*) definida en [179] para transmitir, simular y seleccionar entre comandos de velocidad candidatos basados en una función de coste que combina la distancia a los obstáculos, la distancia a la trayectoria del planificador global y la velocidad a la cual se desplaza el robot. Todo ello configurable según lo exija cada aplicación.

7.1.2 Demostración Experimental

Se realizaron distintas pruebas experimentales de las cuales caben destacar las aplicadas sobre el robot omnidireccional Robotino y sobre el robot diferencial Summit XL.

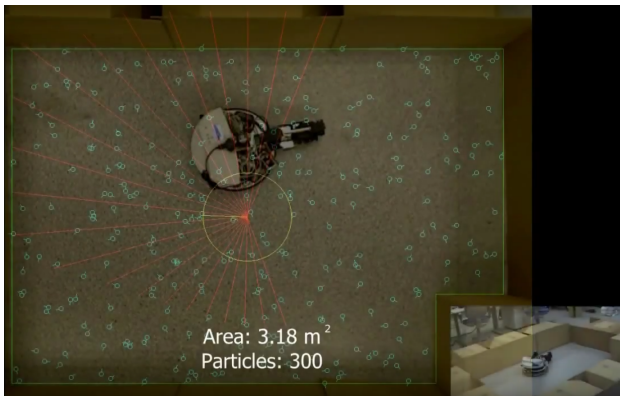
7.1.2.1 Pruebas con Robotino

En primer lugar se implementó el algoritmo de localización por Montecarlo en entornos de interiores tal y como se ha descrito en la sección 7.1.1.2 y en segundo lugar, la navegación autónoma descrita en 7.1.1.3.

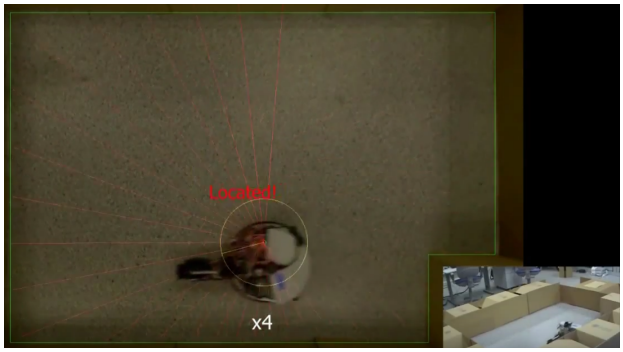
Localización por Montecarlo

En el primer experimento, se diseñó un escenario artificial con cajas de cartón de un área aproximada de 3 m^2, se calculó el mapa de manera manual midiendo con una cinta métrica y se realizaron pruebas con distintos números de partículas, 100, 200 y 300.

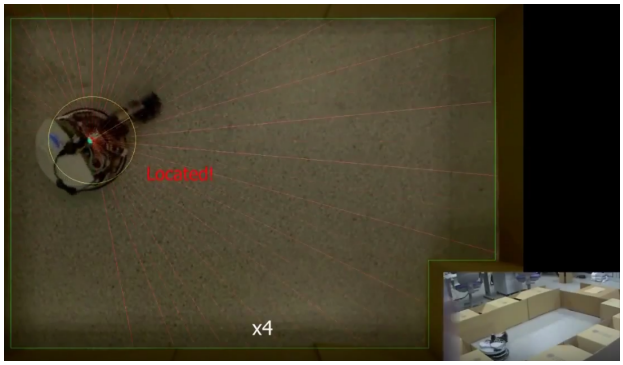
Como se puede apreciar en la sucesión de imágenes de la figura 7.10, en el estado inicial 7.10(a), las partículas se encuentran distribuidas de manera aleatoria dentro del mapa cerrado. Tras algunos desplazamientos del robot (figura 7.10(b)), el algoritmo logra una primera aproximación de la posición del robot gracias al filtrado de las partículas y la correspondencia de las medidas del láser con el perímetro de la estructura del mapa. A medida que se continúa iterando y desplazando el robot, las partículas pueden dispersarse debido al ruido gaussiano, sin embargo la fase de actualización de la posición (filtrado de partículas) tiende a agrupar las mejor ponderadas alrededor de la posición candidata con mejor probabilidad de acierto.



(a) MCL escenario simple fase inicialización



(b) MCL escenario simple fase localización



(c) MCL escenario simple fase robustez

Figura 7.10: Desempeño del algoritmo de localización Montecarlo en Robotino en un escenario simple.

El siguiente experimento se realizó con un mapa parcial de los pasillos del departamento. Este mapa se construyó previamente mediante el algoritmo de SLAM descrito en la sección 7.1.1.1, sin embargo no se realizó de manera autónoma dentro del robot debido a que su capacidad de cómputo no lo permitió. Para obtenerlo, primero se realizó con Robotino una exploración controlada del entorno mientras al mismo tiempo se almacenaba internamente cada posición nueva calculada por la odometría junto con las medidas correspondientes del láser en ese instante. Posteriormente, en un computador externo se procesaron los datos y se obtuvo el mapa 2D requerido por el algoritmo.

En esta segunda prueba, el área abarcó 33 m^2 aproximadamente y se emplearon 400, 500 y 600 partículas para el cálculo de la posición del robot en distintas ejecuciones. Del mismo modo que en el experimento anterior, la serie de figuras 7.11 muestra una secuencia de su ejecución.

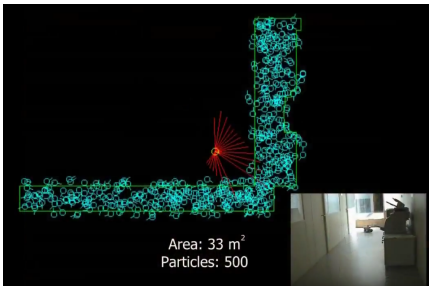
En el estado inicial (figura 7.11(a)), las partículas se esparcen de manera uniforme por el interior del mapa. Según se desplaza el robot por el escenario, muchas de las partículas salen fuera del mapa debido al desplazamiento del robot (figura 7.11(b)). El filtrado de las partículas consigue una primera aproximación de la posición del robot tal y como muestra la figura 7.11(c). No obstante, la aproximación no es buena y pronto se produce una gran dispersión de las partículas como se observa en las figuras 7.11(d) y 7.11(e). Finalmente, el algoritmo converge en su mejor estimación (7.11(f)), posicionando al robot con un error menor a 5 *cm*.

La tabla 7.1 muestra los resultados de las distintas pruebas realizadas con un número de partículas diferentes para los dos escenarios planteados.

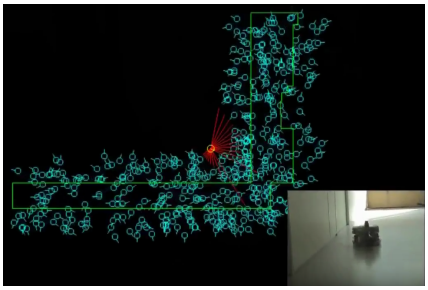
Área del mapa (m^2)	Nº de part.	T./iteración (ms)	T. 1º aprox. (min:sec)	T. mejor aprox. (min:sec)
3.18	100	24	1:03	2:51
3.18	200	59	0:43	1:40
3.18	300	97	0:25	0:25
33	400	141	2:52	>5:00
33	500	195	2:01	3:21
33	600	256	1:23	2:00

Tabla 7.1: Resultados de las pruebas experimentales del *Amcl* ejecutado en Robotino.

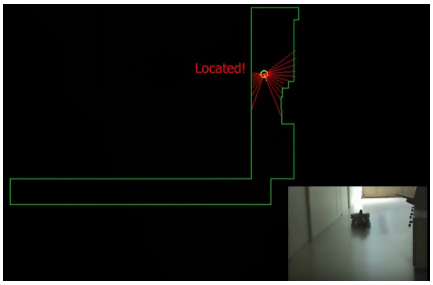
La tabla 7.1 especifica el tiempo por iteración del *Amcl* el cual cabe destacar que superaba el tiempo del bucle de control a partir de los 100 *ms*. Además, se detallan los tiempos de la primera aproximación definida como el instante en el que se obtiene un error de posicionamiento menor a 10 *cm* y la mejor aproximación obtenida con un error de posicionamiento menor a 5 *cm*.



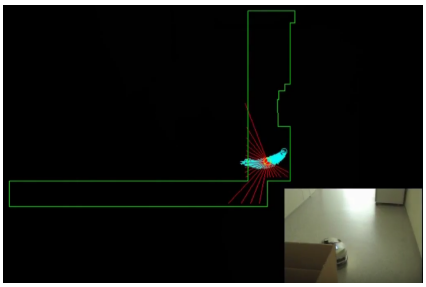
(a) MCL escenario complejo, inicialización



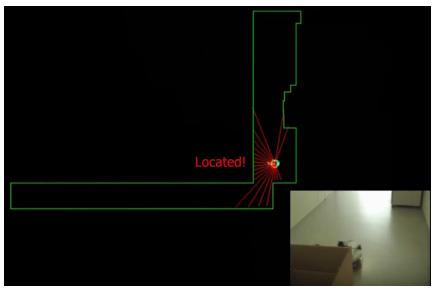
(b) MCL escenario complejo, expansión de partículas



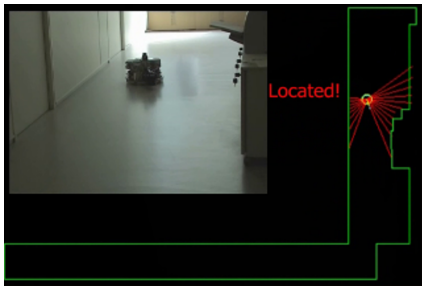
(c) MCL escenario complejo, primera aproximación



(d) MCL escenario complejo, pequeña dispersión



(e) MCL escenario complejo segunda aproximación



(f) MCL escenario complejo segunda aproximación

Figura 7.11: Desempeño del algoritmo de localización Montecarlo en Robotino en un escenario simple.

Hay que mencionar que las trayectorias realizadas durante la localización, son también un factor decisivo en la rapidez de la convergencia del algoritmo, así como la complejidad de la estructura geométrica del mapa conocido. Cuanto menos uniforme sea el mapa, (conteniendo más esquinas o más elementos identificativos) menor ambigüedad se crea en la correspondencia de las partículas y por tanto mayor rapidez de convergencia. Del mismo modo, si las trayectorias no incluyen la detección de dichos elementos identificativos se obtiene un alto grado de ambigüedad en la correspondencia de las medidas y por tanto la convergencia es más lenta.

En último lugar, se puso a prueba la robustez del algoritmo mediante la presencia de outliers dinámicos en el escenario complejo de 33 33 m^2 de área. En estos experimentos se partía de un instante donde el algoritmo había obtenido una primera aproximación con un error menor a 5 *cm* sin ningún tipo de outlier. A continuación, una persona entraba dentro del escenario y caminaba alrededor del robot mientras éste navegaba por el entorno intentando posicionarse. Naturalmente, la detección de la posición de las piernas de la persona generaba una ambigüedad que desembocaba en una dispersión de las partículas solución del algoritmo, como se observa en la figura 7.12(a), ya que dichas posiciones nunca coincidían con el perímetro del mapa. No obstante, bajo esta situación el error de la aproximación de la posición del robot no superó los 15 *cm* en ningún caso y además, el algoritmo recuperaba rápidamente su precisión cuando el outlier desaparecía, como se muestra en la figura 7.12(b).

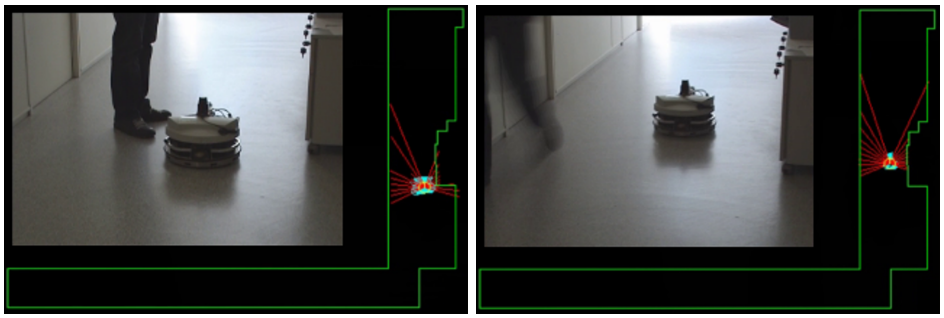


Figura 7.12: Prueba de robustez del algoritmo de localización Montecarlo en Robotino en un escenario complejo.

En la siguiente dirección puede encontrarse un vídeo demostrativos de los experimentos realizados para la implementación del algoritmo MCL con Robotino: `/https://www.youtube.com/watch?v=aIz3EU9MA7U` o también capturando el código QR.

Navegación Autónoma

Se implementó el algoritmo de navegación autónoma descrito en la sección 7.1.1.3 en el robot Robotino. Para obtener una buena ejecución, hubo que establecer la velocidad máxima del robot en 0.1 ms aproximadamente ya que de otro modo la ejecución no era fluida y en ocasiones el robot no se comportaba adecuadamente. Su capacidad de procesamiento limitaba la ejecución en paralelo de la obtención de las medidas del láser (módulo $Hokuyo$) junto con el seguimiento de la trayectoria (módulo $Local_Planner$) por lo que a una velocidad de 0.5 ms, el $Local_Planner$ no siempre actualizaba el mapa local a tiempo provocando que el robot pudiese colisionar con los objetos no reflejados en el mapa global estático. Como medida de seguridad frente a este problema (además de reducir la velocidad de navegación), se redujo aún más la velocidad en los instantes en los que la trayectoria del robot pasaba muy cerca de los objetos detectados o de su propagación. Este fenómeno se observa claramente en el minuto 0:27 del vídeo del experimento.

Para realizar la prueba se colocaron cajas de cartón por el escenario a modo de obstáculos y se posicionó a Robotino dentro del mapa global obtenido del laboratorio. Una vez posicionado, se seleccionó mediante la interfaz de *rviz* la postura destino deseada (flecha de color verde mostrada en la figura 7.13) y se ejecutó el algoritmo de manera satisfactoria. Primero obteniendo la trayectoria y después siguiéndola consiguiendo alcanzar el destino evitando colisionar con los obstáculos.

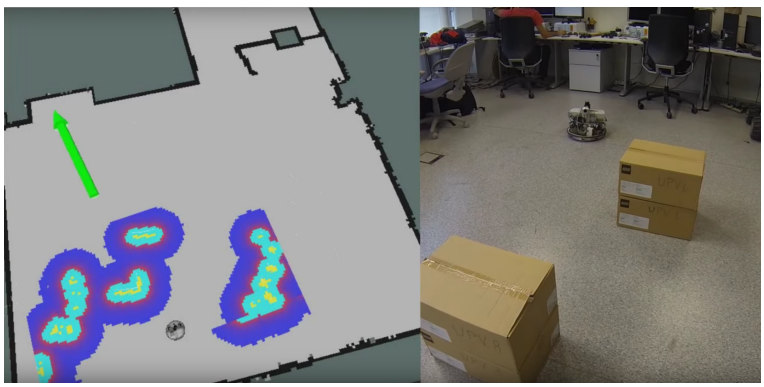


Figura 7.13: Navegación autónoma con Robotino.

En la siguiente dirección puede verse un vídeo del experimento donde se ha divido la imagen. A la izquierda del vídeo puede observarse la interfaz de *rviz* y a la derecha lo que ocurre en el escenario real. La dirección es: `/https://www.youtube.com/watch?v=aAx74qg1AAQ`. También se puede acceder al vídeo mediante la captura del código QR de la derecha.

7.1.2.2 Pruebas con Summit XL

Para realizar otra validación del algoritmo de navegación propuesto, se llevó a cabo su implementación en el robot diferencial Summit XL el cual ofrece unas prestaciones y una capacidad de cómputo mayores que las de Robotino, por lo que la velocidad de navegación se pudo aumentar hasta 0.5 ms sin llegar a presentar ningún problema en tiempos de ejecución.

Después de diversas pruebas de navegación local y global en el laboratorio, se realizó una prueba final en un garaje interior de automóviles, con resultados satisfactorios. Para dicha prueba, primero se escaneó el garaje en cuestión con un sensor láser 3D FARO Focus3D tal y como se expuso en la sección 7.1.1.2 del presente capítulo.

Para la demostración experimental con el Summit XL, se definió la misión de realizar un recorrido cíclico determinado alrededor de un garaje de turismos a modo de centinela de vigilancia.

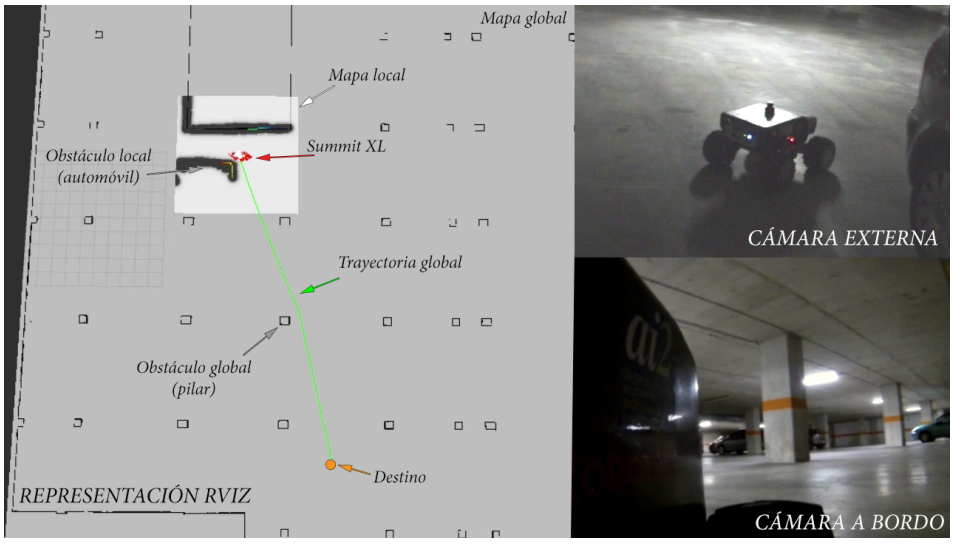


Figura 7.14: Captura de la ejecución del algoritmo de navegación autónoma.

La figura 7.14 muestra un instante de la ejecución del algoritmo. La figura está dividida en tres imágenes, la imagen de la izquierda se corresponde con la representación en la interfaz de *rviz* donde se muestra el mapa global del garaje con paredes y pilares sobre el que está representada la posición del robot, la zona de detección del mapa local de costes (cuadrado en blanco) y la trayectoria global que debe seguir para alcanzar el punto objetivo (línea verde). Además se observa en color rojo, cerca de la posición del robot, las partículas candidatas resultantes de la ejecución del módulo *Amcl* para la localización del robot.

Las imágenes de la derecha de la figura 7.14 se corresponden a dos cámaras, una a bordo del Summit (imagen inferior) y otra externa (superior), las cuales no intervienen para nada en el desarrollo del algoritmo, tal sólo se usaron para registrar su correcto funcionamiento. La demostración completa de las distintas pruebas del experimento está disponible en el siguiente enlace: `/https://www.youtube.com/watch?v=xbinMjXaF4g`, accesible también mediante la captura del código QR de la derecha.

Lo más destacable de estos experimentos es observar la replanificación de la trayectoria del módulo de *Local_Planner* cuando mediante el sensor láser se observa que existen obstáculos (coches) que no figuran en el mapa global y que por tanto el módulo $Global_Planner$ no tiene en cuenta cuando genera la primera trayectoria global hasta el punto objetivo. El planificador local modifica correctamente en tiempo real la trayectoria a seguir e incluso reduce la velocidad en instantes críticos como por ejemplo el ocurrido en el instante 2:00 *min* del vídeo, cuando el robot debe pasar por el estrecho espacio que queda entre dos coches aparcados.

La tabla 7.2 recoge el coste o complejidad computacional de cada una de las fases del algoritmo *amcl*, donde N es el número de partículas utilizadas y M el tamaño de la rejilla del mapa local. En la mayoría de casos en los que se produce el remuestreo del mapa, el procesador del Summit XL alcanza más del 95 % de carga llegando a saturar al 100 % la CPU en momentos críticos. Frente a este problema, se implementó un protocolo de seguridad el cual intervenía en el módulo *Controller* cuando el test de requerimiento de remuestreo daba positivo. En ese caso el robot se quedaba estático en un punto hasta que la tarea de remuestreo terminaba y se realizaba la replanificación de la ruta a seguir.

Fase del algoritmo	Complejidad
Cómputo de la distribución de las partículas	$O(N)$
Actualización del mapa local	$O(N)$
Reasignación de pesos	$O(N)$
Test de Requerimiento de remuestreo	$O(N)$
Remuestreo del mapa	$O(NM)$

Tabla 7.2: Complejidad computacional del *amcl*.

Con estos experimentos queda validado el algoritmo propuesto de navegación autónoma basado en sistemas distribuidos para robots móviles, habiendo conseguido resultados de ejecución satisfactorios para robots con distintas configuraciones y prestaciones, tal y como se ha demostrado.

7.2 Algoritmo de Coordinación para la Navegación de Robots Móviles de Recursos Limitados

En la sección anterior se ha trabajado con algoritmos de alto coste computacional los cuales, a pesar de funcionar correctamente, consumen prácticamente el 100 % del tiempo de procesador de los robots. Esto provoca que difícilmente se puedan integrar nuevos procesos para llevar a cabo estrategias de coordinación de navegación entre agentes que compitan por la ocupación del procesador lo más mínimo, ya que eso desembocaría en que alguno de los módulos críticos de la navegación autónoma no cumpla con sus tiempos de ciclo, haciendo peligrar la estabilidad de la ejecución del algoritmo. Además, la coordinación se pretende realizar mediante el uso de comunicaciones por lo que la gestión de las mismas requiere cierta prioridad dentro del sistema.

Por este motivo y para dar lugar a una posible gestión de la coordinación entre robots o agentes, en esta sección se ha replanteado y reducido al mínimo el coste computacional del algoritmo de navegación de los robots.

El nuevo módulo de navegación definido no conoce ningún tipo de mapa previo ni pretende construirlo, simplemente actúa a dos niveles distintos de ejecución según dos comportamientos definidos:

- A nivel alto, el módulo de navegación espera recibir una postura destino. En caso de hacerlo, ejecuta un algoritmo de seguimiento de trayectoria basado en una estrategia de persecución pura. Cuando el destino final es alcanzado, el procedimiento queda pausado en la pila a la espera de recibir una nueva posición destino. Mientras tanto, el módulo mantiene el avance del robot en línea recta a una velocidad constante.
- A nivel bajo, el módulo de navegación ejecuta un algoritmo Braitenberg el cual actúa en caso de detectar un obstáculo con una probabilidad potencial de colisión, modificando las acciones de la estrategia de seguimiento de trayectoria de nivel alto. Es decir, el comportamiento Braitenberg es más prioritario que el módulo de navegación de nivel alto. En caso de que el robot haya alcanzado un destino solicitado y no haya recibido nuevos destinos, el módulo de navegación lo mantiene navegando en línea recta hasta que encuentre algún obstáculo y el Braitenberg modifique su dirección.

Esta metodología requiere menor capacidad de cómputo y es aplicable prácticamente a cualquier robot de recursos limitados dotado con una mínima instrumentación para realizar la detección de colisiones.

Bien es cierto que cuando se trabaja con robots de recursos limitados, en muchas ocasiones éstos no son capaces de incorporar sensores como los de barrido láser utilizados en la sección anterior. En el caso de el robot e-puck (descrito en la

sección 7.2.2.2) por ejemplo, el rango de detección de los sensores que incorpora no alcanza los 2 *cm* de distancia por lo que la coordinación social entre robots de estas características tiene muy poco margen de actuación, dado que el robot debe comenzar a evitar el obstáculo nada más detectarlo. En tal caso existen dos alternativas, que el vehículo se mueva extremadamente lento para aumentar el margen de detección (solución que no sirve para sensores de resolución binaria) o aumentar de algún modo el rango de detección de obstáculos en plataformas de recursos limitados. Aumentando el rango de detección, se obtendrá mayor tiempo de maniobra para aplicar los algoritmos propuestos en la sección 8.2.3, por lo que, sobre esta última idea se presenta la siguiente sección de esta tesis, la cual expone la arquitectura desarrollada para poder aplicar estrategias de coordinación en plataformas de recursos limitados.

7.2.1 Descripción de la Arquitectura Distribuida Propuesta

Cuando más de un robot comparte el mismo escenario es esencial algún instrumento de detección de objetos para poder evitar colisiones entre los mismos. Como ya se ha comentado, los robots de recursos limitados no suelen ser capaces de incorporar sensores como los rangefinder o sensores de barrido láser, cuya resolución ofrece mejores resultados que los sensores direccionales de infrarrojos o ultrasonidos.

Con robots de recursos limitados como el LEGO Mindstorms NXT (sección 3.3.1) ocurre que no existen sensores de barrido que permitan llevar a cabo una detección fiable y precisa que aseguren la detección de cualquier objeto que se encuentre alrededor. Con sensores de distancia direccionales que sólo focalizan la detección en un limitado ángulo, es difícil que no exista alrededor del robot ningún punto muerto sin detección.

Del mismo modo, el robot mencionado anteriormente, e-puck (descrito en la sección 7.2.2.2), posee un claro ejemplo de sensorización direccional reducida. El anillo de 8 sensores que lo rodea tiene un rango de detección de poco más de centímetro y medio en el mejor de los casos, lo cual ofrece muy poco margen de maniobra ante la detección de un obstáculo.

Sin embargo, para la ejecución de un algoritmo como el Braitenberg a bajo nivel, esta sensorización es suficiente para evitar colisionar en la mayoría de casos de inminente choque. En la siguiente dirección se encuentra un vídeo de la integración de unos sensores de distancia de la marca SHARK con el robot LEGO Mindstorms NXT: `/https://www.youtube.com/watch?v=mlDj2Fand-8` (También disponible mediante la captura del código QR).

El funcionamiento del algoritmo Braitenberg se define en [27]. Concretamente cada sensor embarcado en el robot está ponderado con pesos positivos o negativos los

cuales actúan en cada una de las ruedas según la posición de los sensores. Por ejemplo, los sensores de la izquierda del robot, generan una aceleración positiva de la rueda izquierda y a la vez una aceleración negativa (retroceso) en la rueda derecha, pudiendo hacer girar al robot sobre su propio eje para evitar la colisión.

Dentro de la estructura de módulos o comportamientos que definen los procesos del sistema distribuido, este algoritmo entra en ejecución cuando ocurre el evento de detección de objeto por parte de los sensores físicos o cuando no se tiene un destino que alcanzar. Mientras tanto, en caso contrario, este comportamiento permanece pausado en la pila de procesos.

El problema surge cuando se pretende llevar a cabo una coordinación de navegación entre distintos robots que comparten escenario y dichas plataformas no disponen de una instrumentación que permite detectar al resto sin necesidad de actuar inmediatamente con el Braitenberg para evitar una colisión. La solución radica en aumentar este rango de detección de los robots con recursos limitados para obtener mayor margen de tiempo y así poder ofrecer nuevas soluciones a un nivel de actuación superior al del algoritmo Braitenberg. Para ello se ha desarrollado un agente supervisor gracias al cual es posible llevarlo a cabo tal y como se expone a continuación.

Ampliación del Rango de Detección de los Robots mediante Agente Supervisor

Se ha definido dentro del sistema un supervisor denominado $agente - monitor$ (AMO), el cual gobierna el escenario de pruebas mediante el procesamiento de una cámara cenital instalada en el techo. Este agente único es el encargado de detectar las *poses* de los robots a partir de unos señuelos triangulares de colores instalados físicamente encima de ellos. Cada señuelo tiene un color que identifica cada robot, resolviendo así el problema de *matching* entre robots.

Además, cada *agente-robótico* (AR) crea al inicializarse un agente software denominado $agente - manager$ (AM) quien ejecuta principalmente dos tipos de comportamiento: Uno denominado $conversar$, el cual se utiliza para gestionar todas las comunicaciones entre agentes (consultas, negociaciones, etc) y otro denominado $detectar$, el cual recibe del AMO las posturas de todos los robots detectados y analiza si existe alguna detección entre él y algún otro robot. Cada AM define internamente el radio de detección de cada robot al que representa, pudiéndose modificar su valor en tiempo real.

Por otra parte, el AMO ejecuta un sólo comportamiento en el cual captura, procesa y publica la información para que sea accesible por todos los AM. Este tipo de mensaje se compone de una secuencia de la tupla, *<código de color>,<postura>*. Cada agente *virtual* AM lee dicha información y conoce el color del AR a quien

representa por lo que simplemente calcula la distancia euclídea entre su posición y la del resto y comprueba si es menor que el rango de detección. De esta manera es como se le otorga al robot la capacidad indirecta de ser capaz de detectar objetos a distancias que físicamente por sus limitaciones no puede. Además, no se informa directamente al *AR* de ello, si no que es únicamente el *AM* quien tiene esa información. Esto permite aislar al *AR* de comunicaciones innecesarias con otros agentes que puedan desembocar en retardos no deseados; ya que este agente ejecutado localmente por el robot, tiene principalmente dos comportamientos críticos ya descritos, el comportamiento *Braitenberg* y el comportamiento *Navegación*, encargados de la evasión de obstáculos y el control de navegación.

En la figura 7.15 se puede observar la estructura de la arquitectura propuesta en función de los comportamientos que ejecuta cada agente y las interacciones entre ellos.

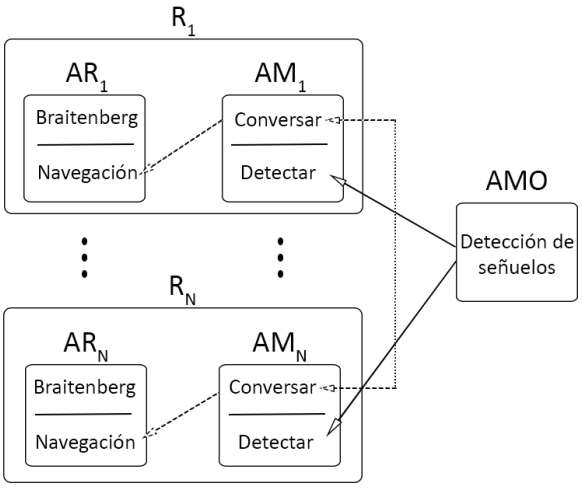


Figura 7.15: Relación entre comportamientos y sus comunicaciones

El agente *AMO* envía las posiciones detectadas al módulo *Detectar* de cada AM_i. En caso de detectar una amenaza, dicho módulo lanzará el comportamiento de *Conversar* el cual gestionará la solución de las posibles colisiones con los agentes *AM* de los robots implicados y finalmente,la solución derivará en una posición destino nueva, la cual se le enviará al comportamiento *Navegación* de los AR_i implicados.

Conviene mencionar que el *AM* se ha definido como un agente con movilidad basado en [40] por lo que puede cambiar su ubicación de ejecución en caso de necesitarlo. Para la arquitectura propuesta, todos los *AM* se ejecutan en computadores

conectados físicamente a una misma red tal y como muestra la figura 7.16, la cual se corresponde con el esquema de la experimentación con robots e-pucks descrita más adelante.

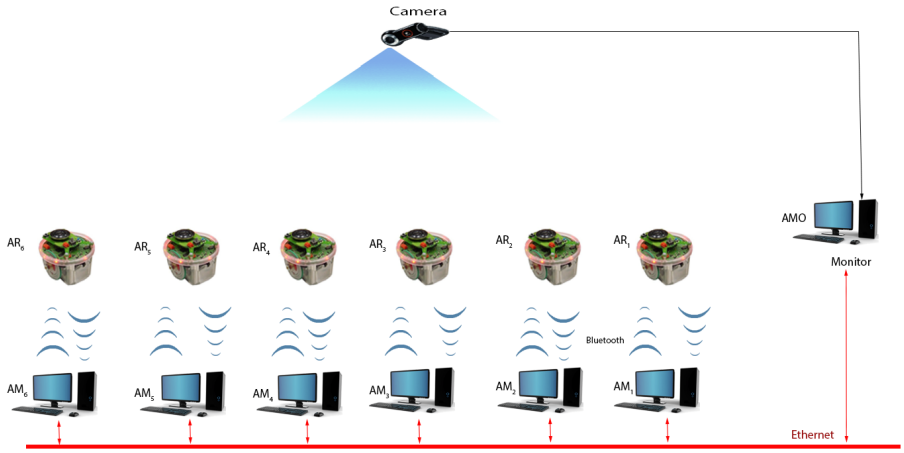


Figura 7.16: Arquitectura desarrollada para la ampliación del rango de detección de robots

Este planteamiento se debe a que el canal de comunicación inalámbrico más crítico (más lento) del sistema es el que conecta los robots móviles con los computadores, (comunicación Bluetooth en el peor de los casos) por lo que se ha intentado reducir al máximo el número de conversaciones a través de ese canal. El resto de comunicaciones, las conversaciones entre el AMO y los AM o las conversaciones entre los distintos AM, se realizan mediante Ethernet por lo que los retardos son del orden de ms.

Expuesta la arquitectura desarrollada, a continuación se describen las pruebas realizadas para demostrar su funcionamiento.

7.2.2 Demostración Experimental: Agrupaciones Holónicas de Robots

En esta sección se describe el trabajo implementado para una prueba experimental basada en la coordinación de robots móviles para la creación de agrupaciones holónicas. El trabajo se llevó a cabo en dos fases. Una primera fase de simulación donde se comprobó el buen funcionamiento del mismo y una segunda fase donde se trasladó la ejecución a un marco experimental con un grupo de 6 robots e-pucks.

7.2.2.1 Experimentos en Plataforma de Simulación

En simulación, la estructura de la arquitectura cambia mínimamente. El agente *AMO* se traduce a un agente *IGU* encargado de gestionar la interfaz o la arena de simulación como si fuese el resultado de las capturas de una cámara cenital. Además, los robots reales son sustituidos por agentes software que emulan sus mismos comportamientos tal y como se ha explicado en la sección 6.3.1.

Bajo la arquitectura descrita en la sección 7.2.1, se plantea una aplicación de agrupaciones de robots donde cada uno persigue individualmente a un líder en base al valor de un índice de valoración.

El marco de esta prueba parte del descrito en la sección 6.1.1, donde además de otorgar a los agentes la misión de localizarse entre sí mediante el intercambio de mensajes cuándo su *consciencia* de mal posicionamiento lo requiere(véase la sección 5.2.2), los robots deben intentar perseguir al robot detectado con mejor índice de valoración (líder).

El intercambio de mensajes se realiza también del mismo modo que en la sección 6.1, sólo que ahora se añade un parámetro más dentro de cada mensaje, el cual corresponde al índice de valoración. En un instante t un R_i detecta a un grupo de robots a una distancia menor que D_i, el robot detectado que presente mejor índice de valoración, será el elegido por R_i como líder y se le añadirá la misión de perseguirlo e intentar no perderlo de vista.

En el contexto descrito en 7.2.1, es el agente *AM* quien selecciona al líder y quien envía la posición del líder como nuevo destino del comportamiento de *navegación* al *AR* a quien representa. Si la distancia al líder supera D_i en algún momento, el *AM* seleccionará a otro robot como nuevo líder en caso de existir uno que cumpla la condición de tener mayor índice de valoración. En caso contrario, el *AM* no seleccionará ningún destino nuevo y el robot seguirá una línea recta a velocidad constante con el algoritmo Braitenberg ejecutándose a bajo nivel. De este modo se van creando agrupaciones de robots dinámicas basadas en jerarquías temporales de igual a igual, es decir, holarquías.

Para definir este índice de valoración se consideraron dos opciones:

- **Un valor constante asignado a cada robot.** Este índice puede representar la puntuación resultante de un conjunto de medidas de las prestaciones del robot como su capacidad de cómputo, la fiabilidad, el alcance y la calidad de sus sensores, por ejemplo.
- **La inversa del valor medio de la covarianza de cada robot.** Aprovechando los resultados del capítulo 5, el índice se define para cada instante t como la inversa del valor medio del error de estimación de posición $P_{k,p}$, (elipsoides 3σ, ecuación (5.7)) .

Índice fijo

En la secuencia de la imagen 7.17 se puede observar un ejemplo de la ejecución del experimento con 10 robots en simulación cuando el índice de valoración del líder se define como un valor constante para cada robot.

La interfaz mantiene la misma representación descrita en la secciones 6.3.1 y 6.2, observándose visualmente la evolución de la covarianza de cada robot en tiempo real como un círculo interno centrado en su posición. Además se ha añadido en la representación, el índice asignado de cada robot, el número identificativo de cada uno y a priori, se ha asignado un color propio para cada robot. Sin embargo, cuando un robot encuentra un líder, toma el color del líder con un tono más oscuro y no vuelve a su color original hasta que no deja de ser subordinado de alguien. De esta manera se identifican visualmente los grupos de robots en el escenario y su evolución.

La imagen 7.17(a) describe una situación con 5 holarquías. La mayor está formada por los 5 robots de color turquesa donde el robot 7 con un índice de 39 es el líder de los robots 1,2,3,8 con índices 35, 21, 12 y 29, respectivamente (ya que $39 > 35, 21, 12, 29$). Si se observa con atención, se muestra que el robot 2, quien también forma parte del grupo, no sigue al líder global de la holarquía, sino que su líder es el robot 1 de índice 35 ya que su rango de detección no llega a alcanzar al robot 7. Por otra parte se muestran 4 holarquías individuales, donde ningún robot tiene asignado ningún líder. El robot 10 detecta al 38 que tiene un índice de 23 y al ser menor, no ejerce de líder. Además, se observa también que los robots 4, 5 y 9 han sobrepasado el valor del evento que provoca una solicitud de actualización de posición (expresado por los signos de exclamación y el círculo interno de color oscuro).

Tras unos instantes, en la figura 7.17(b) el número de holarquías se ha reducido a 4 por el hecho de que el robot 4 ha detectado al robot 10 quien ha comenzado a ejercer como líder de una nueva holarquía al no detectar a ningún otro robot de mayor índice dentro de D_4. Además esta detección ha permitido al robot 4 actualizar su posición por lo que el valor del error de estimación de posición se ha reducido hasta el punto de no necesitar que lo posicionen.

En las figuras 7.17(c), 7.17(d) y 7.17(e) se puede observar la evolución de las 3 holarquías resultantes hasta ese momento. El robot 9 detecta al robot 5 como líder, cambia su color a un tono oscuro del líder y ambos se actualizan las posiciones manteniéndose bien localizados entre ellos. Avanzando en el tiempo hasta la figura 7.17(f), se muestra la agrupación de los robots en únicamente 2 holarquías cuyos líderes son el robot 7 con un índice de 39 y el robot 10 con un índice de 38.

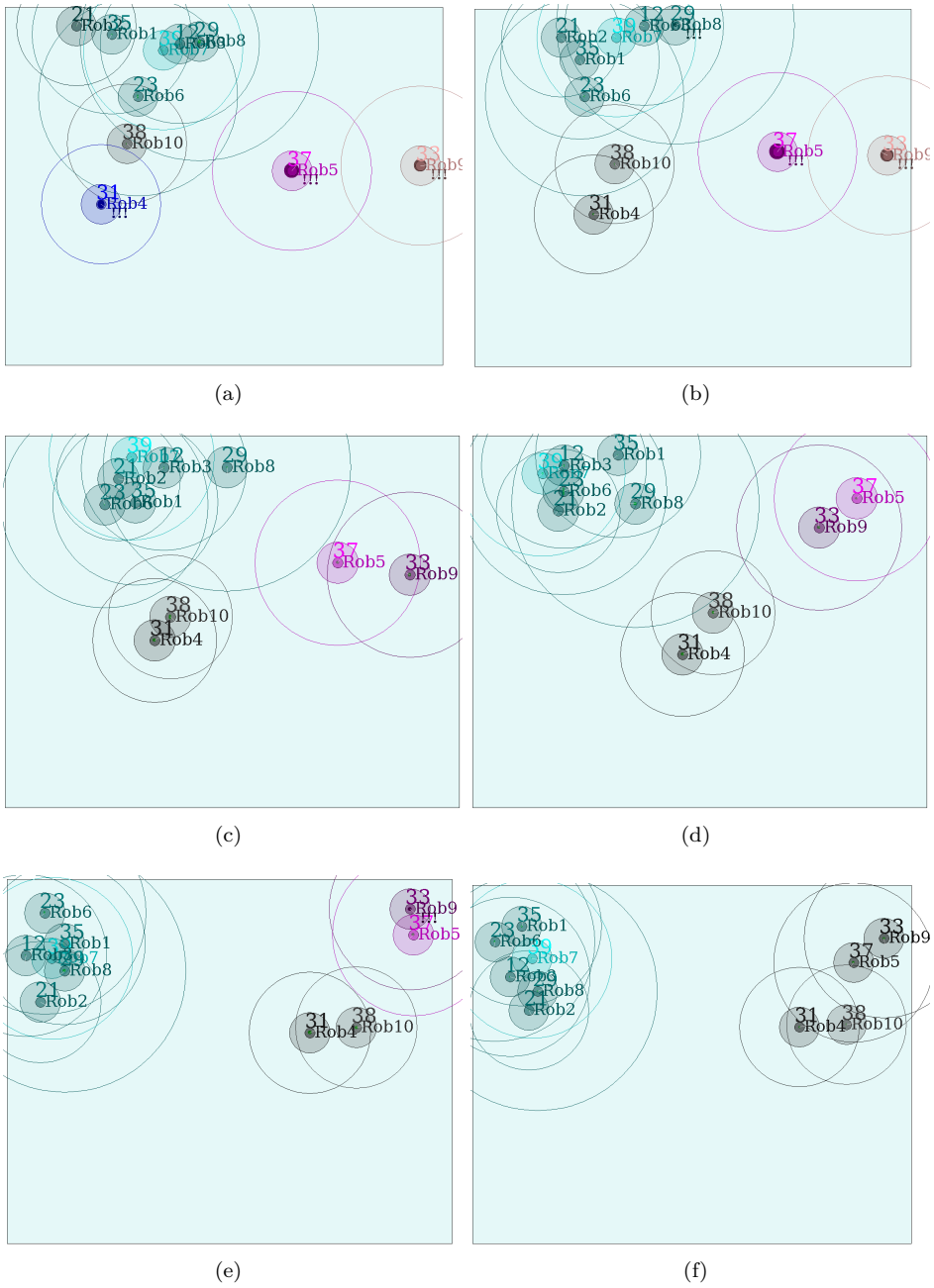


Figura 7.17: Secuencia del algoritmo de agrupaciones holónicas (índice fijo).

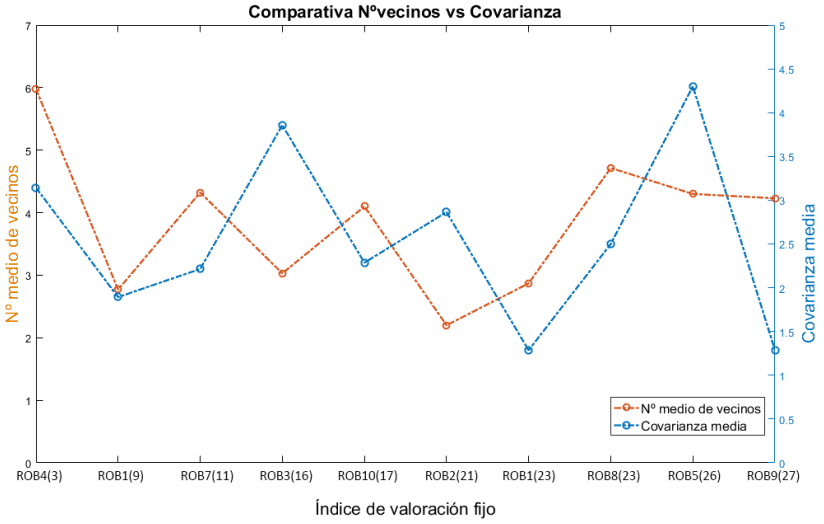


Figura 7.18: Gráfico del nº medio de vecinos y la covarianza media en una prueba de 10 minutos de duración con índice fijo.

La gráfica 7.18 muestra los resultados de una ejecución del experimento de una duración de 10 *min*. En el eje X se refleja el índice de cada robot en orden ascendente, el eje Y de la izquierda refleja el número medio de vecinos detectados durante toda la ejecución, y el derecho, la covarianza media de cada robot. Como puede observarse no existe una relación directa ni significativa entre el índice asignado a cada robot, el número medio de vecinos y la covarianza. Esto es debido a que tener un buen índice de valoración no implica tener una covarianza baja.

Índice dinámico

Del mismo modo, la figura 7.19 muestra una secuencia de un ejemplo de ejecución del algoritmo con el índice de valoración dinámico, definido como la inversa de la covarianza del error de posicionamiento de cada iteración.

En la figura 7.19(a) se observa 3 holarquías, una individual formada por el robot 3 quien no consigue detectar a ningún vecino, otra holarquía de color azul liderada por el robot 4 con un índice de 5 y una tercera holarquía liderada por el robot 10 (el mejor posicionado en ese instante) con un índice de 7.
La imagen 7.17(b) muestra cómo el robot 3 ha conseguido detectar al robot 10, quien sigue ejerciendo de líder con un índice igual a 5. Sin embargo, en la holarquía inferior ahora el robot 9 es quien ejerce de líder sobre el resto con un índice de 3.
En la secuencia de las imágenes 7.17(c),7.17(d) y 7.17(e) puede observarse la variación entre los índices de posicionamiento y los cambios de líder de cada holarquía.

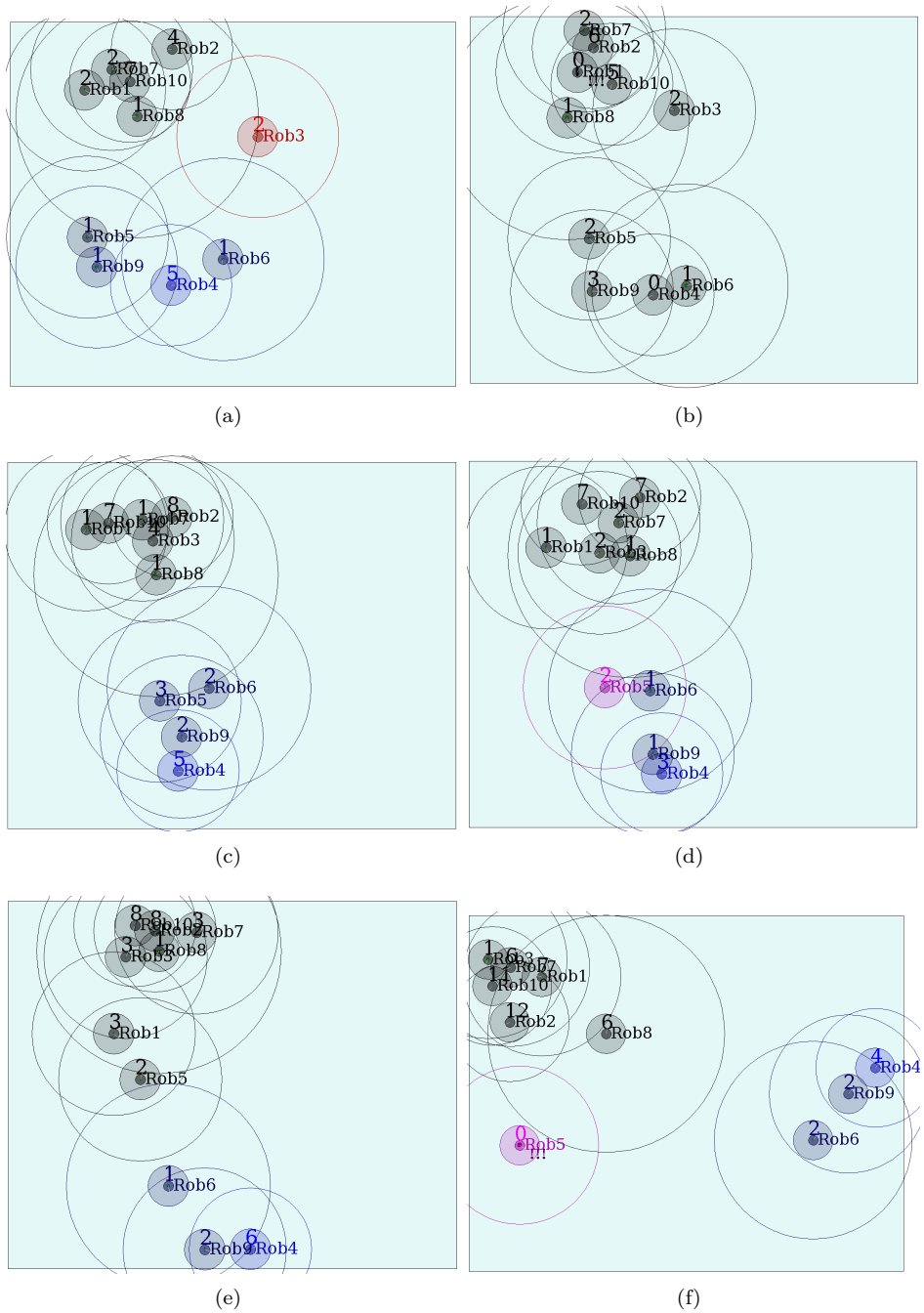


Figura 7.19: Secuencia del algoritmo de agrupaciones holónicas (índice dinámico).

Conforme pasa el tiempo es posible encontrar situaciones como la del robot 5 en la figura 7.17(f) quien se aleja del resto de robots al no encontrar un líder, con un índice reducido a 0 y buscando un vecino con quien actualizar su posición.

La gráfica de la figura 7.20 muestra la evolución de la covarianza de 10 robots en una prueba de 10 minutos de duración, comparada con la evolución media del número de robots detectados en cada momento.

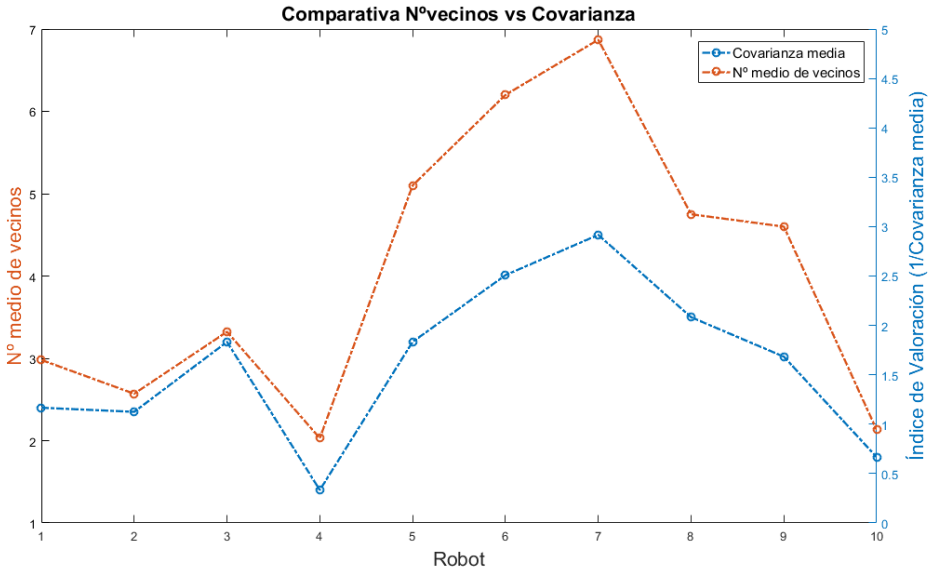


Figura 7.20: Gráfico del nº medio de vecinos y la covarianza media en una prueba de 10 minutos de duración con índice dinámico.

Al diferencia de la gráfica 7.18, en esta gráfica se confirma que la relación entre el número de vecinos y la covarianza conseguida es sin duda significativa, dado que confirma el hecho de que cuanto más tiempo permanecen en holarquías los robots, mejor es la estimación de su posición gracias a las actualizaciones de sus vecinos. Además, los robots que detectan a un mayor número de vecinos, obtienen el índice de valoración más alto por lo que se premia también el tamaño del grupo. Se concluye pues que a pesar de que cada robot actúa de manera individual, en conjunto se promueve la unión del grupo en beneficio de todos.

En el siguiente enlace se muestra un vídeo del experimento con los dos índices propuestos: `/https://www.youtube.com/watch?v=jWwDItV2wwo`.

7.2.2.2 Plataforma Experimental: e-puck

El e-puck (Figura 7.21) es un robot móvil de ruedas diferencial desarrollado por el Dr. Francesco Mondada y Michael Bonani en 2006 en la EPFL, (Instituto Federal Suizo de Tecnología de Lausanne). Es un robot de arquitectura hardware y software abierta que se compone de dos ruedas motorizados con capacidad de girar en direcciones opuestas para cambiar la dirección del robot. Además cuenta con varios sensores, de los cuales cabe destacar los 8 infrarrojos incorporados, útiles para medir la distancia a los objetos cercanos o al menos detectar su presencia. También tiene un módulo de comunicaciones Bluetooth versión 1.2, gracias al cual se ha podido integrar dentro del canal de comunicaciones multi-agente. Las señales de activación de los motores paso a paso están disponibles y pueden ser utilizados en lugar de los codificadores incrementales (ya que físicamente no están disponibles en los e-puck).

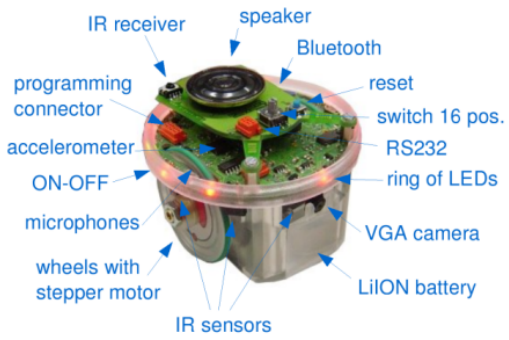


Figura 7.21: Robot móvil diferencial e-puck

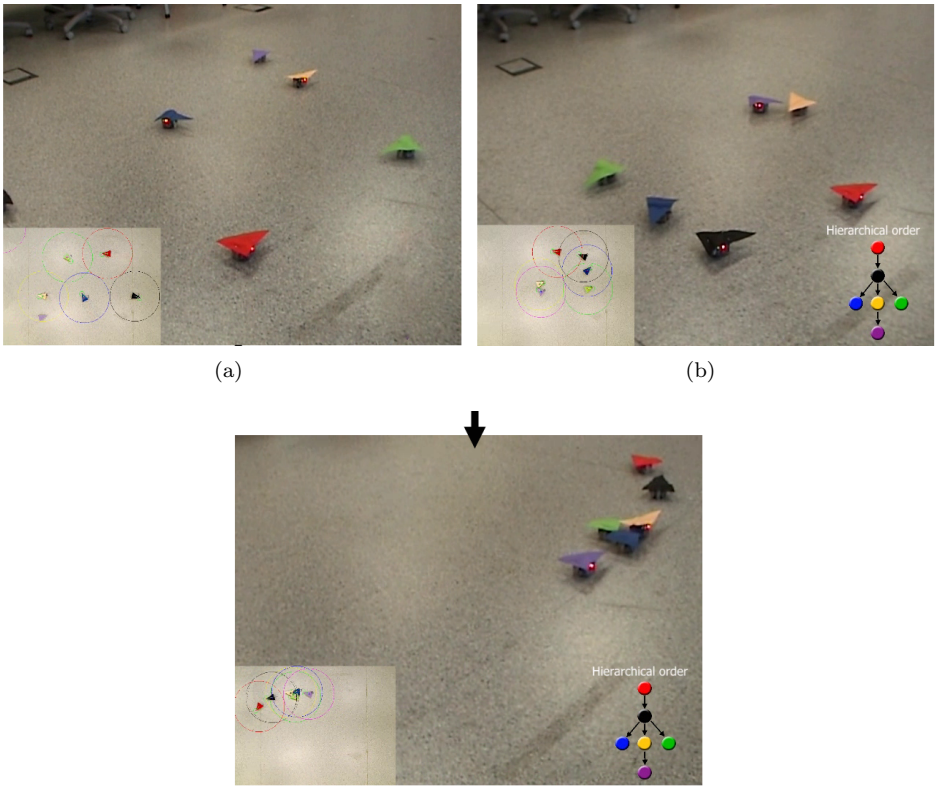
La unidad de control del robot está basada en un microcontrolador dsPIC de Microchip que combina un procesador de 16 bits con una unidad de procesamiento digital de señales (DSP), proporcionando un procesamiento de datos muy eficiente. El microcontrolador dispone de 8kB de memoria RAM y 144kB de memoria flash. Funciona a una frecuencia de 64 MHz y proporciona hasta 16 MIPS.

Para la programación de las rutinas de control del robot se utiliza el sistema operativo de tiempo real y un entorno de desarrollo. El sistema operativo Erika Enterprise, que es un sistema operativo de código abierto OSEK/VDX. El entorno de desarrollo es la herramienta de diseño de sistemas embebidos RT-Druid. Éste, a su vez se sirve de MPLAB para la compilación y descarga de las aplicaciones en el e-puck.

7.2.2.3 Experimentos con Agrupaciones de Robots e-pucks

El mismo experimento realizado en simulación, se implementó en robots e-pucks de recursos limitados con la diferencia de que en este caso la jerarquía temporal se creó únicamente en base a un índice numérico aleatorio ya que los robots e-pucks tienen una capacidad de cómputo muy limitada y no permiten las operaciones complejas que requieren los filtros de fusión sensorial para el cálculo de su covarianza.

No obstante, la arquitectura desarrollada se corresponde con la descrita en la imagen 7.16 con la ligera modificación de que en caso de que un robot no encontrase ningún líder a quien perseguir, el robot debía trazar una trayectoria circular de 2 metros de diámetro en lugar de seguir una trayectoria recta a velocidad constante. De este modo se evitó que los robots salieran fuera del rango de detección de la cámara cenital que proporcionaba el posicionamiento.



(a) (b)

(c)

Figura 7.22: Secuencia del algoritmo de agrupaciones dinámicas con robots e-pucks.

Para poder ver el efecto de la agrupación de los robots se partió de una situación donde cada robot se movía de manera aleatoria con el comportamiento Braitenberg (figura 7.22(a)). En un momento dado se establecía la jerarquía temporal y los robots cambiaban a ejecutar el comportamiento de navegación donde deben intentar alcanzar al robot líder (figura 7.22(b)), y finalmente conseguían alinearse y formar una agrupación sólida tal y como muestra la figura 7.22(c).

En el siguiente enlace se muestra un vídeo con diversas pruebas de agrupaciones llevadas a cabo mediante este algoritmo sobre los robots e-puck, con resultados satisfactorios: `/https://www.youtube.com/watch?v=kYJ6a64b4iM`.

La estructura de la arquitectura propuesta resulta adecuada y es aplicable a robots móviles de recursos limitados tal y como ha quedado demostrado.

7.3 Conclusiones del Capítulo

En el presente capítulo se propusieron y desarrollaron diversos algoritmos relacionados con la problemática asociada a la navegación y coordinación de robots móviles con recursos limitados, haciendo uso de metodologías basadas en agentes.

- Se ha aportado un algoritmo de navegación autónoma de vehículos basado en una arquitectura distribuida donde cada nodo o módulo de ejecución se encarga de gestionar procesos independientes entre sí, los cuales mediante una comunicación basada en publicación-suscripción consiguen recopilar información para ejecutar módulos más complejos.

- Se ha descrito un algoritmo de SLAM desarrollado mediante un filtro de partículas basado en la arquitectura *ROS* e implementado en el módulo *Mapping* (Alg. 14 para la obtención de un mapa conocido del entorno por donde se desplaza un robot móvil.

- Se ha presentado un algoritmo de localización basado en Montecarlo e implementado en un nodo denominado *Amcl* (Alg. 15) el cual recibe el mapa obtenido por el módulo *Mapping* y mediante el uso de filtros de partículas y correspondencias entre las medidas del láser y el mapa, posiciona al robot dentro del mapa conocido.

- Se ha detallado el algoritmo desarrollado para la navegación autónoma de vehículos basado en los nodos *Mapping* y *Amcl* descritos previamente. Además se han expuesto los dos planificadores implementados para la navegación en interiores, uno basado en información global obtenida del mapa previo

conocido del entorno y otro basado en la información local obtenida por las medidas del láser (sec. 7.1.1.3).

- Se ha mostrado una demostración experimental de la localización del algoritmo *Amcl* implementada sobre el robot omnidireccional de recursos limitados Robotino, con resultados satisfactorios incluso cuando intervienen outliers durante su ejecución (sec. 7.1.2.1).

- Se ha presentado los resultados de la implementación y ejecución del algoritmo de navegación autónoma propuesto sobre los robots Robotino y Summit XL en distintos escenarios de interiores ofreciendo resultados satisfactorios en tiempos de cómputo y ejecución. Además se presentan diversos vídeos demostrativos de las pruebas realizadas.

- Se ha descrito la metodología utilizada para dotar de mayor y mejor sensorización a robots móviles de recursos limitados mediante el uso de MAS (sec.7.2.1).

- A continuación se ha presentado la arquitectura definida del modelo de agentes utilizado para desarrollar un algoritmo de agrupaciones holónicas mediante la coordinación y navegación de robots móviles basado en la arquitectura MAS.

- Se ha implementado dicha metodología y ejecutado primeramente en demostradores de simulación y posteriormente en las plataformas de recursos limitados *e-pucks*; ofreciendo buenos resultados y consolidando la buena ejecución del algoritmo de coordinación.

Capítulo 8

Detección y Evasión de Colisiones de Robots Mediante Consenso entre Agentes

La capacidad de un robot para detectar y evitar colisiones forma parte del paso previo al problema de planificación de movimientos tal y como se justifica en trabajos como [42] o [175].

En los algoritmos anteriormente expuestos, se ha implementado una evasión de obstáculos a través de dos metodologías distintas: replanificando las trayectorias (como ocurría en la navegación con el robot Summit XL), o mediante un algoritmo reactivo de expulsión como el Braitenberg (utilizado en el capítulo anterior). En esta sección se propone un algoritmo de detección y evasión de colisiones de alto nivel basado en consensos entre agentes conseguidos a través de conversaciones entre los mismos.

El algoritmo propuesto se sitúa en un nivel por encima de algoritmos reactivos como el Braitenberg sin ser excluyentes entre sí. En una determinada situación es posible que trabaje uno u otro según lo crítica que resulte la detección del obstáculo.

En el presente capítulo se realiza una breve exposición de las metodologías más relevantes sobre evasión de colisiones utilizadas en robótica móvil, después se describe la metodología propuesta para el algoritmo de detección y evasión de colisiones y, finalmente se presentan los resultados experimentales bajo los cuales se valida el algoritmo.

8.1 Estrategias de Detección y Evasión de Colisiones

Una colisión se puede definir como la situación en la que dos o más cuerpos ocupan el mismo espacio en un mismo instante de tiempo.

Para simplificar el cálculo de posibles colisiones, muchos métodos de la literatura consideran ciertas premisas como se expone en [85], de las cuales las más populares se exponen a continuación:

- **Volúmenes envolventes.** Una de las técnicas más utilizadas para reducir las iteraciones del cálculo de las colisiones es realizar una simplificación del volumen del cuerpo de los objetos como se expone en trabajos como [98], [164] o [88]. En robótica suele ser habitual simplificar la forma geométrica de los robots ya que normalmente el modelo fiel a la realidad contiene un gran número de vértices y aristas. Al compartir la superficie por la que se desplazan, en la robótica móvil normalmente no se tiene en cuenta la componente Z del modelo, por lo que la mayoría de robots se reducen a modelos 2D simplificados en un círculo o un cuadrilátero.

- **Descomposición espacial.** Hacer uso de un sistema que descomponga el espacio de trabajo en diversas zonas ofrece la ventaja de no tener la obligación de recorrer todo el espacio entre trayectorias para calcular si puede ocurrir una colisión o no. Gracias a esta discretización utilizada en trabajos como [105] o [176], el método de detección tan sólo debe comprobar si hay o no colisión en aquellos espacios donde los dos cuerpos se aproximan o ocupan un mismo espacio discretizado, descartando aquellas zonas del espacio donde no existe riesgo de colisión.

Por otro lado, en cuanto a los métodos de detección de colisiones, principalmente destacan los siguientes:

- **Algoritmo de las características más cercanas** Este método parte de la base de que a cada instante de tiempo los objetos a estudiar se mueven y pueden rotar. Dependiendo de las características geométricas del objeto esto puede ser determinante o no porque por ejemplo, en el caso de que ambos objetos sean circunferencias 2D como suele ocurrir en robótica móvil, la rotación del objeto no resulta relevante. Sin embargo, para modelos más complejos poco uniformes, las rotaciones pueden ser críticas para el cálculo de distancias entre los objetos. Para evitar este problema, esta estrategia parte del cálculo de la distancia entre otras características definidas. El algoritmo que adopta esta idea realiza una serie de iteraciones en las que comprueba si el par de características estudiadas cumplen el teorema que determina si existe colisión. En caso negativo, no existe intersección y se devuelve el par de características más cercanas. En el caso de que no se cumplan las condiciones, se actualiza a unas nuevas características y se continua iterando

hasta encontrar las características más cercanas que comparten los dos objetos. [25] y [177] son ejemplos de trabajos en los que se utiliza esta metodología.

- **Método jerárquico de detección de colisiones** En trabajos como [88] o [99], se construye una jerarquía en forma de árbol para n volúmenes envolventes de tal forma que un método iterativo lo recorre buscando la mejor solución. Los volúmenes envolventes más simples forman los nodos de las hojas del árbol jerarquizado. En un nivel superior se agrupan estos nodos en un nuevo volumen de envolvente de manera recursiva hasta que se obtiene un volumen de envolvente que agrupa a los n objetos. Esta jerarquía permite comprobar desde la raíz si existe colisión entre los nodos más altos. Sólo en caso de existir colisión, el algoritmo desciende a los nodos inferiores comprobando de nuevo si existe colisión entre los nodos hijos y así sucesivamente. De este modo se reduce la complejidad de los cálculos a la representación de la envolvente máxima que genera la colisión.

Las metodologías no son excluyentes entre sí por lo que algunos trabajos combinan varias metodologías como [88] o [197]. De hecho, el algoritmo propuesto también utiliza una combinación de las estrategias de recubrimiento de envolventes y de la descomposición espacial como se expone más adelante.

Una vez determinada y confirmada la detección de la colisión, el siguiente paso es actuar mediante alguna respuesta para evitarla. En el caso de la robótica móvil existen dos alternativas normalmente: modificar la velocidad o la dinámica de los vehículos que intervienen en el caso de que los dos estén en movimiento o replanificar sus trayectorias para que no colisionen entre ellos.

Los métodos que modifican la dinámica y velocidad de los robots suelen englobarse dentro de las estrategias continuas de detección de colisiones (CCD, Continuous Collision Detection) donde cada periodo de muestreo se actúa según la detección del entorno percibida; mientras que los métodos de replanificación de trayectorias suelen pertenecer a estrategias discretas de detección de colisiones (DCD, Discrete Collision Detection), donde dada una detección de colisión, se lleva a cabo una replanificación de la trayectoria la cual debería asegurar que no volviera a ocurrir otra detección de colisión. Ambas metodologías son compatibles entre sí, pero trabajan a distinto nivel de actuación.

La replanificación de trayectorias responde a problemas geométricos de alto nivel como el bordeo de obstáculos expuesto en [108], la planificación mediante algoritmos A* como en [189] o la planificación mediante gráficos de visibilidad como en [165].

Las estrategias CCD se corresponden con algoritmos reactivos de bajo nivel lo cuales no requieren una sensorización compleja como los expuestos en [143], [144] o [199]. Sin embargo, el algoritmo reactivo más popular y extendido en el campo de la robótica móvil es sin duda el Braitenberg [27].

Algunos trabajos del campo de la robótica móvil han basado sus estrategias de evasión de colisiones en algoritmos reactivos CCD, ofreciendo resultados de evasiones óptimos. Es decir, que los robots evitan su colisión modificando lo mínimo posible su trayectoria. Es el caso de los algoritmos *ORCA* [185], [3] o *DRCA* [95], [96] donde la metodología utilizada se basa en ir modificando mínimamente la velocidad y los ángulos de giro de los robots según perciben las detecciones a los objetos cercanos. Además no utilizan ningún tipo de comunicación entre robots y el mismo algoritmo se ejecuta de manera individual en cada robot lo cual garantiza que el sistema es totalmente descentralizado. Sin embargo y como consecuencia, una de sus principales limitaciones recae sobre el hecho de que todos los algoritmos anteriores requieren reciprocidad para funcionar correctamente y suponen que mediante la instrumentación de los robots, es posible calcular la velocidad lineal y angular de los objetos detectados, una premisa poco plausible para robots de recursos limitados o sensores de detección con resolución binaria. Además, su intención de buscar la optimalidad de la evasión de manera reactiva, los sitúa al mismo nivel bajo de ejecución que el algoritmo Braitenberg, por lo que difícilmente son compatibles bajo una misma ejecución.

Por otro lado, existen pocos trabajos que empleen el uso de las comunicaciones para ofrecer estrategias de evasión de colisiones. La mayor parte de trabajos se centran en la problemática conversacional de llevar a cabo el consenso entre agentes para evadir una colisión o para generar formaciones, como son el caso de [64] o [13].

El algoritmo de evasión de colisiones propuesto en la siguiente sección utiliza las comunicaciones para conseguir información adicional con la que poder ofrecer una solución óptima a todos los robots que intervienen en dicha colisión. Además es aplicable a robots de recursos limitados y puede responder a cualquier nivel de ejecución, siendo compatible con otros métodos CCD como el Braitenberg o el *ORCA* cuando sólo se aplica sobre el nivel más alto de ejecución.

8.2 Descripción del Algoritmo Propuesto de Detección de Colisiones

Naturalmente, cuanto mayor es el rango de detección de los sensores, antes se detecta un obstáculo, por lo que existe un mayor margen de maniobra para evitar la colisión con el mismo y por tanto más posibilidades de desarrollar nuevos algoritmos de evasión.

El alcance de las comunicaciones inalámbricas supera el rango de cualquier sensorización embarcada en robots móviles en la mayoría de los casos. Este hecho se ve acentuado en casos prácticos como el robot e-puck que posee unos sensores de distancia con un rango máximo (D_r) menor a 2 cm y sin embargo el alcance de sus comunicaciones Bluetooth (C_r) permite alcanzar los 10 m de distancia de

nodo a nodo. Esta premisa justifica el uso de las comunicaciones para el cálculo de estrategias de evasión de colisiones, previas a una detección física entre dos objetos.

La habilidad comunicativa permite llevar a cabo conversaciones, negociaciones y acuerdos entre agentes a un nivel superior a los algoritmos individuales. En este trabajo se ha aportado dicha habilidad de comunicación para combinarla con la capacidad de detección de objetos ampliada descrita en la sección 7.2.1; consiguiendo de esta manera ampliar el tiempo de actuación de la evasión de colisiones en robots de recursos limitados, utilizando información adicional proveniente de las comunicaciones y de ese modo obtener una solución alternativa y complementaria a las propuestas hasta ahora para realizar una evasión más *inteligente*. Por este motivo la metodología engloba tres conceptos concretos de tres grandes áreas distintas de trabajo: la detección y evasión de obstáculos, los sistemas multi-agente y la inteligencia artificial como método de negociación entre agentes.

8.2.1 Marco de Aplicación

Para el desarrollo del algoritmo propuesto se ha aplicado la metodología de los volúmenes envolventes para simplificar los cálculos de detección de colisiones. Todos los robots se representan geométricamente como círculos los cuales engloban la figura del robot en un plano 2D donde una intersección entre sus perímetros o una penetración entre sus círculos suponen una colisión entre robots.

En este trabajo cada robot i se identifica por una letra del alfabeto A, B, ..., Z y su modelo en cada instante t se define según su radio r_i y su postura $P_i(t) = \{x_i(t),y_i(t),\theta_i(t)\}$ definida respecto a los ejes de referencia global (X_G,Y_G), los cuales son comunes para todos los miembros del grupo, G_R. Bajo estos conceptos, dadas las posturas $P_A(t) = \{x_A(t), y_A(t), \theta_A(t)\}$ y $P_B(t) = \{x_B(t), y_B(t), \theta_B(t)\}$ de los robots R_A y R_B, la inecuación 8.1 define si existe o no colisión para el instante (t).

$$(r_A + r_B) \leq \sqrt{(x_B(t) - x_A(t))^2 + (y_B(t) - y_A(t))^2} \tag{8.1}$$

En caso de incumplirse la inecuación existirá colisión en el instante t siempre que los robots se encuentren en esas posiciones.

Dado que se pretende aplicar el algoritmo a robots móviles, lo habitual es que cada robot esté cumpliendo alguna misión que le obligue a realizar un desplazamiento por el espacio XY del escenario. En conceptos de optimalidad, el desplazamiento más corto entre dos puntos siempre es la línea recta por lo que una buena premisa desde la que comenzar a resolver la problemática, es la de asumir un escenario en el que todos los robots realizan trayectorias en línea recta entre un punto inicial y un punto final.

De esta forma, para el instante t_s se puede definir la postura que ocupa el robot en ese momento como $P_i(t_s) \in \mathbb{R}^2$. También es posible añadir la especificación de la velocidad v_i a la que el robot se mueve o pretende moverse para realizar la trayectoria, ya que en t_s el robot puede encontrarse quieto o en movimiento.

Bajo estas premisas el reto inicial al que se enfrenta el algoritmo es a la determinación de si dadas las trayectorias de dos robots en un mismo escenario, es posible que los robots colisionen o no.

8.2.2 Obtención del Instante de Máxima Aproximación

En lugar de ofrecer la resolución binaria de si existe o no colisión, el algoritmo propuesto basa la detección de colisión en el cálculo del instante de máxima aproximación que existe entre las trayectorias, tal y como se describe a continuación.

Conociendo la posición inicial P_i para los robots R_A y R_B en el instante inicial t_s y asumiendo que ambos van a seguir una trayectoria en línea recta a una velocidad v_i constante, es posible obtener una posición final o destino para un tiempo t_g^i definida como $P_i(t_g^i) \forall R_i$ tal y como muestra la figura 8.1.

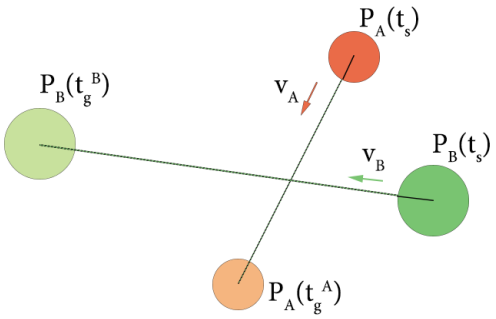


Figura 8.1: Esquema de trayectorias rectilíneas de los robots R_A (naranja) y R_B (verde).

Para aplicar el algoritmo, en el siguiente paso es necesario hacer coincidir los tiempos t_g^A y t_g^B con el objetivo de parametrizar las trayectorias como se explica más adelante. Es evidente que difícilmente se va a cumplir que los dos robots alcancen sus destinos en el mismo instante (lo cual cumpliría $t_g^A{=}t_g^B$), por lo que para conseguir esta igualdad se selecciona el menor tiempo de los dos y se calcula la posición destino intermedia donde el otro robot se encontrará en el instante de menor t_g^i.

Por ejemplo, dada la figura 8.1, si por simplificación se supone que los dos robots se desplazan a la misma velocidad ($v_A{=}v_B$), es evidente que R_A alcanzará antes su destino que R_B, o sea que $t_g^A < t_g^B$. Se selecciona pues t_g^A como el instante final t_g

a estudiar (t_g^A=t_g). Al tratarse de trayectorias en línea recta, es posible calcular la posición $P_B(t_g^A)$ o, lo que es lo mismo, $P_B(t_g)$, interpolando sobre la trayectoria. De este modo se obtienen las posiciones finales $P_A(t_g)$ y $P_B(t_g)$ las cuales se muestran en la figura 8.2.

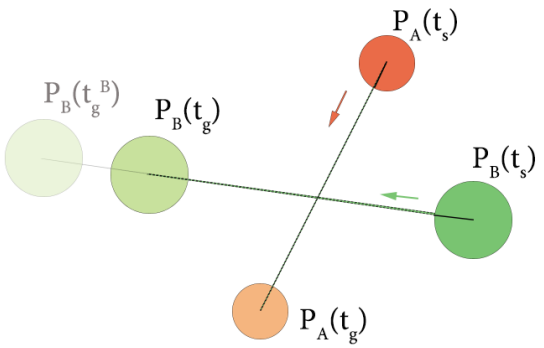


Figura 8.2: Obtención de las posiciones de R_A y R_B en el instante t_g común

Cabe destacar que la asunción v_A=v_B no es estrictamente necesaria. En caso de no cumplirse basta con calcular t_g^A y t_g^B y seleccionar el menor valor entre ellos, lo cual es precisamente cómo se realiza este cálculo en los experimentos prácticos.

Llegado este punto, se parametriza cada trayectoria de los robots mediante λ para obtener todas las infinitas posiciones intermedias entre $P_i(t_s)$ y $P_i(t_g)$ cuando $\lambda \in [0,1]$. La ecuación 8.2 define dicha parametrización para $R_i(\lambda)$

$$R_i(\lambda) = \{(P_i(\lambda), r_A) : P_i(\lambda) = P_i(t_s) + \lambda \cdot (P_i(t_g) - P_i(t_s)); \\ t = t_s + \lambda(t_g - t_s); \forall \lambda \in [0,1]\} \tag{8.2}$$

Con esta parametrización se cumple que $R_i(\lambda)$ y $R_i(t)$ con $t = t_s + \lambda(t_g - t_s)$ son exactamente iguales para todo $t \in [t_s, t_g]$ y $\forall \lambda \in [0,1]$

Además, observando la ecuación 8.2 y dados R_A y R_B, se puede encontrar el instante de máxima aproximación t_M calculando el valor de $\lambda_c \in [0,1]$ que minimiza la ecuación 8.3.

$$min\left\{||P_A(\lambda) - P_B(\lambda)|| - (r_A + r_B)\right\}; \forall \lambda \in [0,1]\} \tag{8.3}$$

En realidad, $||P_A(\lambda) - P_B(\lambda)||$ puede interpretarse como un segmento en el espacio de Minkowski [57], cuyos extremos se definen como $P_s^M = P_A(t_s) - P_B(t_s)$ y $P_g^M = P_A(t_g) - P_B(t_g)$. En este segmento se calcula λ_c definida como la posición en el espacio de Minkowski, $P_A(\lambda_c) - P_B(\lambda_c)$, más cercana al origen de coordenadas

O. Este origen representa la igualdad $P_A(\lambda) = P_B(\lambda)$ en el espacio euclídeo, lo que implica una penetración del 100 % entre los dos robots.

Para obtener $\lambda_c \in [0, 1]$ hay que proyectar O en el segmento, $O^{\perp}$, tal y como se describe en la ecuación 8.4, donde $\lambda_c \in [0, 1]$.

$$\begin{aligned} \lambda_c &= -\frac{P_s^M \cdot (P_g^M - P_s^M)}{||P_g^M - P_s^M||^2} \\ \therefore O^{\perp} &= P_s^M + \lambda_c \cdot (P_g^M - P_s^M) \end{aligned} \tag{8.4}$$

La imagen 8.3 muestra un ejemplo en el espacio de Minkowski donde puede observarse que el segmento $||P_A(\lambda) - P_B(\lambda)||$ pasa por el origen de coordenadas por lo que existirá colisión entre R_A y R_B como se determina más adelante.

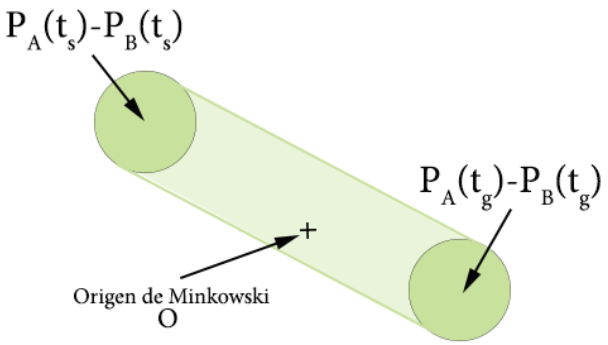


Figura 8.3: Obtención de la trayectoria en el espacio de Minkowski para los robots R_A y R_B. Se producirá colisión ya que la trayectoria pasa por el origen de coordenadas

El valor de λ_c debe cumplir estrictamente pertenecer a un rango de valores entre 0 y 1. El caso para $\lambda_c > 1$ significa que el momento de máxima aproximación entre las trayectorias se producirá después de t_g, en cuyo caso que habría que comprobar expresamente si para $\lambda_c = 1$ existe colisión; y si $\lambda_c < 0$, quiere decir que el momento de máxima aproximación ocurriría en un instante anterior a t_s si los dos robots hubieran mantenido el rumbo de ambas trayectorias previamente a t_s.

Una vez obtenida $\lambda_c \in [0, 1]$, se puede obtener la proyección sobre el segmento, $O^{\perp}$, como define la ecuación 8.4 y la distancia d_M como la diferencia entre las dos posiciones, tal y como se muestra la ecuación 8.5. Además, si a esa diferencia se le resta la suma de los radios de los robots (r_A y r_B), se puede deducir que se

producirá colisión siempre que $d_M \leq 0$. Se puede considerar que hay colisión sólo con haber *contacto* ($d_M = 0$).

$$d_M = (P_A(\lambda_c) - P_B(\lambda_c)) - (r_A + r_B) \tag{8.5}$$

Llegado este punto el algoritmo es capaz de conocer si existirá o no colisión entre las dos trayectorias. Además es posible calcular el instante preciso t_M en el que ocurrirá el momento de máxima aproximación como el tiempo inicial más el instante en el que ocurra λ_c, como muestra la ecuación 8.6.

$$t_M = t_s + \lambda_c(t_g - t_s) \tag{8.6}$$

Se puede afirmar entonces que en caso de que $d_M \leq 0$ y $\lambda_c \in [0, 1]$, se cumple que en el instante $t_M \in [t_s, t_g]$ se produce el momento de máxima aproximación entre R_A y R_B con una penetración entre los dos cuerpos igual a $|d_M|$.

En la figura 8.4 se puede observar gráficamente la colisión producida por las posiciones $P_A(t_M)$ y $P_B(t_M)$ en el instante de máxima penetración.

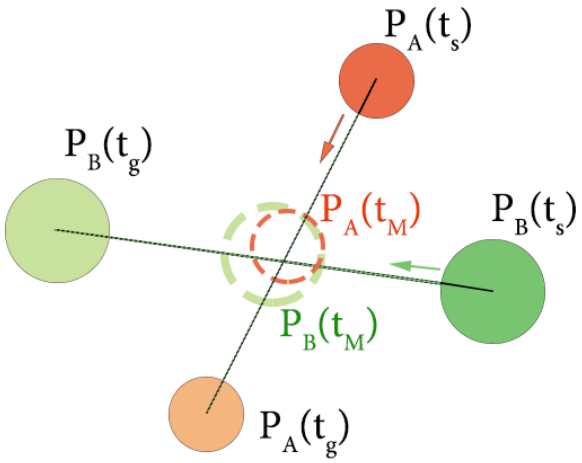


Figura 8.4: Posición de máxima penetración para el instante t_M entre los robots R_A y R_B en el espacio euclídeo.

En este punto el algoritmo conoce si existe colisión o no, y en caso de existir, conoce cuándo se va a producir dicha colisión y dónde. El siguiente problema que se debe abordar consiste en ofrecer opciones de cómo evitar dicha colisión de una manera óptima.

8.2.3 Metodología de Evasión de Colisiones

Conociendo el instante de la colisión t_M y la penetración entre los dos robots en el instante de máxima aproximación, la solución óptima es replanificar las trayectorias de los robots para que en ese instante t_M exista al menos una distancia entre sus centros mayor a la suma de sus radios. La distancia mínima de desplazamiento para conseguirlo es precisamente d_M. Pero además, se puede obtener el vector de desplazamiento óptimo definido como $\hat{v}_{MTD}$, a partir de la ecuación 8.7, teniendo en cuenta que $||\hat{v}_{MTD}|| = 1$.

$$\hat{v}_{MTD} = \frac{O^{\perp}}{||O^{\perp}||} \tag{8.7}$$

La imagen 8.5 muestra gráficamente la correspondencia de las variables d_M, $O^{\perp}$ y $\hat{v}_{MTD}$ en el espacio de Minkowski, donde $O^{\perp}$ se corresponde con la proyección del origen de Minkowski sobre el segmento $||P_A(\lambda) - P_B(\lambda)||$.

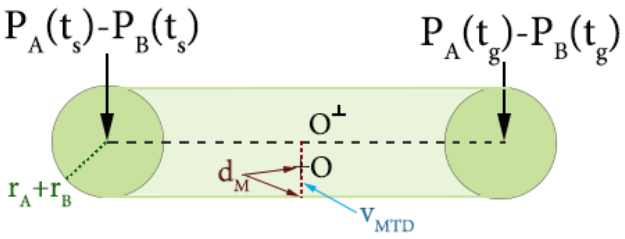


Figura 8.5: Proyección del origen de Minkowski en $||P_A(\lambda) - P_B(\lambda)||$

Para el caso de R_A y R_B se pueden definir dos configuraciones para las posiciones libres de colisión según las ecuaciones 8.8 y 8.9.

$$\begin{aligned} P'_A(t_M) &= P_A(t_M) - \delta \cdot \alpha \cdot d_M \cdot \hat{v}_{MTD} \\ P'_B(t_M) &= P_B(t_M) + \delta \cdot (1-\alpha) \cdot d_M \cdot \hat{v}_{MTD} \end{aligned} \tag{8.8}$$

$$\begin{aligned} P'_A(t_M) &= P_A(t_M) + \delta \cdot \alpha \cdot d_M \cdot \hat{v}_{MTD} \\ P'_B(t_M) &= P_B(t_M) - \delta \cdot (1-\alpha) \cdot d_M \cdot \hat{v}_{MTD} \end{aligned} \tag{8.9}$$

Donde la definición de $P_A(t_M)$ y $P_A(t_M)$ se detalla en la ecuación 8.10.

$$\begin{aligned} P_A(t_M) &= P_A(t_s) + \lambda_c(P_A(t_g) - P_A(t_s) \\ P_B(t_M) &= P_B(t_s) + \lambda_c(P_B(t_g) - P_B(t_s) \end{aligned} \tag{8.10}$$

El parámetro $\delta \geq 1$ se corresponde con un margen de seguridad añadido a la distancia mínima de aproximación, por lo que si $\delta = 1$ existirá contacto entre las posiciones $P'_A(t_M)$ y $P'_B(t - M)$. Además el parámetro $\alpha \in [0, 1]$ configura el porcentaje de evasión de cada uno de los robots ya que para separar las dos posiciones existen infinitas soluciones derivadas de la combinatoria de la participación de cada uno de los robots. Si $\alpha = 1$ o $\alpha = 0$ sólo uno de los robots modificará su trayectoria para evitar la colisión mientras que el otro continuará su trayectoria original.

Esta combinatoria dentro del conjunto de soluciones óptimas es lo que genera la posibilidad de añadir cooperatividad, coordinación y negociación entre los agentes para encontrar una solución colaborativa que resuelva las colisiones de manera óptima.

La imagen 8.6 muestra un ejemplo de una solución óptima con $\alpha = 0,7$, es decir, un robot evita el 70 % (el robot naranja R_A) y el otro un 30 % (robot verde R_B) y con $\delta = 1.03$.

Las nuevas posiciones $P'_A(t_M)$ y $P'_B(t_M)$ generan una desviación de la trayectoria de cada uno de los robots resolviendo la colisión con un margen de distancia de 0.03 (3 %) la suma de sus radios ($\delta = 1.03$). Por este motivo se observa la separación entre las posiciones libres de colisión.

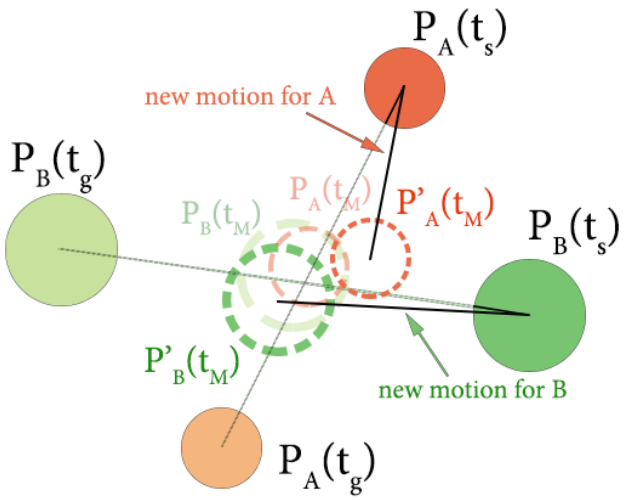


Figura 8.6: Ejemplo de solución para la colisión entre los robots R_A y R_B

Una vez descrito el procedimiento matemático para obtener el conjunto de soluciones de evasión de colisiones ofrecidas por el algoritmo desarrollado, se procede a exponer en detalle la implementación del algoritmo y su estructura a nivel de programación y gestión distribuida.

8.2.4 Estructura de ejecución del Algoritmo

Se considera el marco de un escenario donde varios robots siguen una trayectoria recta entre un punto inicial y un punto final, los cuales se van alternando una vez que son alcanzados por el robot. Todos los robots tienen un agente software representativo en el MARS por lo que se considera que no existe un objeto en movimiento en el escenario que no figure como agente.

La implementación del algoritmo propuesto se lleva a cabo en las seis fases mostradas por la figura 8.7, las cuales se explican a continuación.

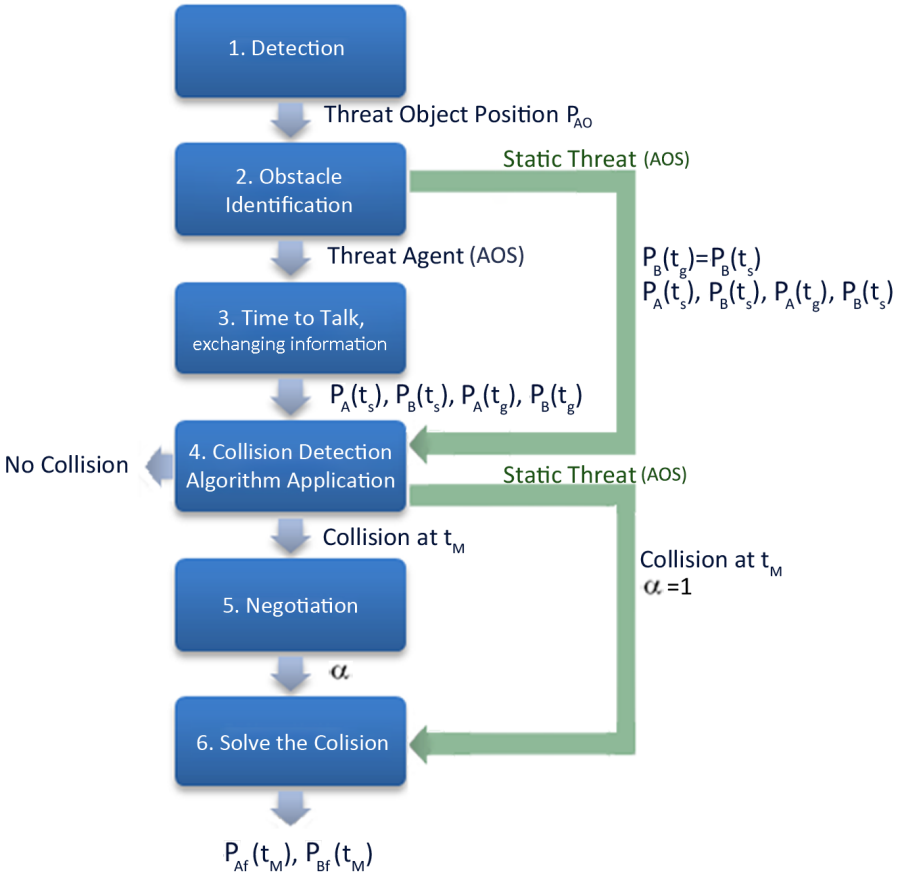


Figura 8.7: Fases de la metodología propuesta para la evasión de colisiones.

1. **Detección.** Cada robot tiene definida un área de detección de objetos. Cuando un robot detecta un obstáculo que puede suponer una amenaza de colisión (a partir de ahora AO), el sistema calcula su posición global P_{AO} y la envía al agente AM que lo representa dentro del MARS.

2. **Identificación del obstáculo.** Cuando un agente software recibe una P_{AO} de su robot, debe identificar qué tipo de amenaza supone: otro agente dinámico o un objeto estático del escenario. Para ello, dicho agente ejecuta el comportamiento *detección*, quien consulta al resto de agentes del sistema quién está situado en el área centrada sobre la posición P_{AO}. En caso de que ningún agente se encuentre dentro de esa área, la amenaza AO se identifica como un objeto estático AO_S y directamente el algoritmo salta a la fase 4. En otro caso, la AO es identificada como otro agente detectado, AO_D y comienza la fase 3.

3. **Conversar e intercambiar información.** Cuando dos agentes se ven involucrados en una posible colisión deben comunicarse entre ellos para obtener la información necesaria para aplicar el algoritmo de detección y evasión de colisiones. Concretamente cuando un AM contesta a otro AM para confirmarle que se encuentra dentro de una posición P_{AO} detectada, le envía directamente sus datos de posición, velocidad, radio y posición final; ya que las entradas al algoritmo son ocho: las posiciones de cada agente involucrado $P_A(t_s)$ y $P_B(t_s)$, las posiciones destino de cada uno $P_A(t_g^A)$ y $P_B(t_g^B)$, sus velocidades v_A y v_B y los radios r_A y r_B. Entre ellos determinan $t_g = min(t_g^A, t_g^B)$ y el agente más lento recalcula su posición para $P_i(t_g)$ obteniendo todo lo necesario para aplicar el algoritmo propuesto en la sección 8.2.2.

4. **Aplicar el algoritmo de detección y evasión de colisiones.** En caso de que en la fase 2, el AO se identificase como AO_S, su posición será siempre la misma para $\forall t$ $(P_B(t_s)=P_B(t_g))$. Obtenidos $P_A(t_s)$, $P_B(t_s)$, $P_A(t_g)$, $P_B(t_g)$, r_A y r_B, se aplica el algoritmo propuesto para conocer si existe colisión en algún instante de la trayectoria. En caso de que no exista colisión, la amenaza se descarta y el método termina. Si existe, el método informa al agente del tiempo t_M en el que se producirá la máxima aproximación.

   El siguiente paso es evitar la colisión mediante el método descrito en la sección 8.2.3. Si el objeto es estático AO_S, el agente que ha detectado la colisión (*detector*) calculará su nueva posición libre de colisión para t_M con $\alpha = 1$, haciéndose cargo del 100 % de la evasión y saltando a la fase 6 del algoritmo. En otro caso, los agentes involucrados deben negociar para consensuar de qué porcentaje de evasión se encarga cada uno.

5. **Negociación** Para decidir el porcentaje de carga de evasión que cada agente debe asumir, los agentes pueden intercambiar cualquier información como la prioridad, el peso de la carga que transportan, la cantidad de combustible o energía que les queda, la dificultad de maniobrabilidad,... O incluso un factor que represente un conjunto de características dispares ponderadas. En este punto es donde es posible aplicar estrategias de IA que ofrezcan soluciones inteligentes según los casos particulares que pueden ocurrir. Una vez que cada agente accede a asumir su α, el agente *detector* ejecuta la última fase del algoritmo.

6. **Resolver la colisión** El agente *detector* ejecuta el método de evasión de colisiones y obtiene dos posiciones nuevas a alcanzar para el instante t_M, $P_{Af}(t_M)$ y $P_{Bf}(t_M)$. El agente *detectado* recibe por parte del *detector* la posición que debe alcanzar y el tiempo en el cual debe hacerlo ($P_B(t_M)$ y t_M respectivamente). Si α no es ni 0 ni 1, ambos robots cambiarán su trayectoria para dirigirse a la nueva posición parcial al mismo tiempo, evitando así que colisionen entre sí. Una vez pasado el instante t_M los robots pueden seguir hasta su destino final, volviendo a ejecutar el algoritmo en caso de volverse a detectar otra amenaza posterior a t_M.

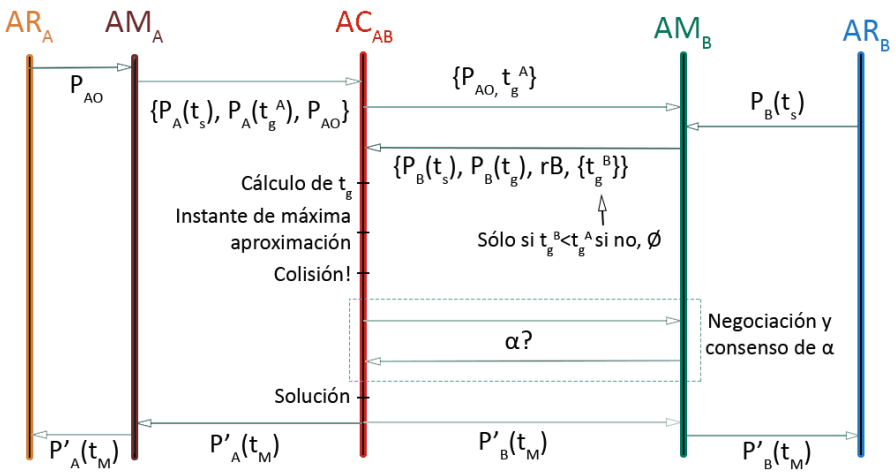


Figura 8.8: Esquema de comunicaciones entre agentes en la ejecución del algoritmo.

En cuanto a la estructura de los módulos de ejecución en la plataforma, cabe destacar que por cada colisión que se detecta, el agente *detector* (AM_A) lanza un nuevo agente *colisión*(AC) encargado específicamente de gestionar dicha colisión. Ese agente $AC_{A,i}$ se ocupa de ejecutar las fases de la 2 a la 6, las cuales incluyen las comunicaciones entre agentes para la recopilación de los parámetros necesarios para el algoritmo.

La primera comunicación entre agentes se produce cuando en la fase 2 se consulta qué agente está cercano a la posición de amenaza detectada P_{AO}. En dicha solicitud, el agente $AC_{A,i}$ envía tanto la posición detectada como el instante en el que alcanzará su destino t_g^A. Frente a esa consulta realizada a modo de propagación, tan sólo contesta el agente AM_i que efectivamente se encuentran cerca de esa posición (se asume R_B, por ejemplo).

Cuando se recibe una contestación, puede recibirse de dos maneras tal y como muestra el esquema de comunicación de la figura 8.8. Esto se debe a que el agente AM_B, conoce t_g^A y t_g^B por lo que si $t_g^B < t_g^A$ (el robot R_B alcanza antes su destino que R_A), informa al $AC_{A,B}$ de su $P_B(ts)$, su r_B, su t_g^B y su $P(t_g^B)$, para que el agente AC interpole la posición final del robot R_A teniendo en cuenta que $t_g = t_g^B$. O también puede ocurrir que $t_g^B > t_g^A$, en cuyo caso el AM_B responde al $AC_{A,B}$ únicamente con $P_B(ts)$, su r_B, y su $P(t_g^A)$, asumiendo de esta forma que $t_g = t_g^A$.

En ese momento el $AC_{A,B}$ calcula si existirá colisión y en caso afirmativo le propondrá el parámetro α al AM_B. En esta fase se produce una negociación determinada para cada caso concreto que interese, que termina con el consenso del parámetro α entre los dos agentes. El agente $AC_{A,B}$ calcula las posiciones finales de cada uno de los robots implicados y se las envía a su AM_i correspondiente, terminando en ese instante su ejecución.

Expuesta de manera detallada la metodología del algoritmo propuesto, cabe mencionar que, tal y como ocurría en la sección 7.2, independientemente de la aplicación del nuevo algoritmo, a bajo nivel cada agente puede ejecutar un algoritmo reactivo como el Braitenberg que actúe sólo cuando la detección es inminente.

El algoritmo 16 muestra el pseudocódigo de los cálculos iterativos necesarios para aplicar la metodología propuesta de evasión de colisiones.

Algoritmo 16: Algoritmo de evasión de colisiones propuesto

Entrada:
$P_A(t_s)$, $P_A(t_g^A)$, r_A, v_A: *Parámetros de* R_A
$P_B(t_s)$, $P_B(t_g^B)$, r_B, v_B: *Parámetros de* R_B
Salida:
$P'_A(t_M)$: *Posición de* R_A *libre de colisión*
$P'_B(t_M)$: *Posición de* R_B *libre de colisión*

//Cálculo de t_g
$t_g = min\left\{t_g^A, t_g^B\right\}$

//Obtención de las posición $P_A(t_g)$ o $P_B(t_g)$
Si $t_g != t_g^A$ **entonces**
  $P_A(t_g) = |P_A(t_s) + t_g * v_A|$
si no
  $P_B(t_g) = |P_B(t_s) + t_g * v_B|$
fin

//Puntos inicial y final del segmento en el espacio de Minkowski
$P_s^M = P_A(t_s) - P_B(t_s)$
$P_g^M = P_A(t_g) - P_B(t_g)$

//cálculo del instante máxima aproximación
$\lambda_c = -\frac{P_s^M \cdot (P_g^M - P_s^M)}{||P_g^M - P_s^M||^2}; \lambda_c \in [0,1]$
$t_M = t_s + \lambda_c \cdot (t_g - t_s)$

//cálculo de la distancia entre los robots en el instante de máxima aproximación
$d_M = (P_A(t_M) - P_B(t_M)) - (r_A + r_B)$

//Si $\exists$ colisión...
Si $d_M \leq 0$ **entonces**
  //Cálculo de la proyección del origen de Minkowski en el segmento
  $O^{\perp} = P_s^M + \lambda_c \cdot (P_g^M - P_s^M$
  $\hat{v}_{MTD} = \frac{O^{\perp}}{||O^{\perp}||}$
  //Cálculo de las posiciones de máxima penetración en el espacio euclídeo
  $P_A(t_M) = P_A(t_s) + \lambda_c(P_A(t_g) - P_A(t_s)$
  $P_B(t_M) = P_B(t_s) + \lambda_c(P_B(t_g) - P_B(t_s)$
  //Cálculo de las nuevas posiciones libres de colisión
  $P'_A(t_M) = P_A(t_M) + \delta \cdot \alpha \cdot d_M \cdot \hat{v}_{MTD}$
  $P'_B(t_M) = P_B(t_M) - \delta \cdot (1 - \alpha) \cdot d_M \cdot \hat{v}_{MTD}$
fin

8.3 Demostración Experimental: Consenso 1 vs 1

Con el objetivo de verificar la efectividad del algoritmo propuesto, se llevó a cabo su implementación mediante simulación, a través de la plataforma descrita en la sección 6.3.1.

Para simular los robots, se ha implementado el modelo dinámico de las aceleraciones de las ruedas del robot de la ecuación (B.57) que se utiliza como entrada al modelo (B.34) (versión 3 partículas) el cual define las entradas del modelo de la postura del robot diferencial de la ecuación (B.24). De este modo, el agente software ejecuta una réplica del algoritmo de control del robot real y, migrar o integrar la plataforma de simulación con robots reales, tan sólo requiere el traslado del código del agente AR_i entre plataformas tal y como se expone más adelante.

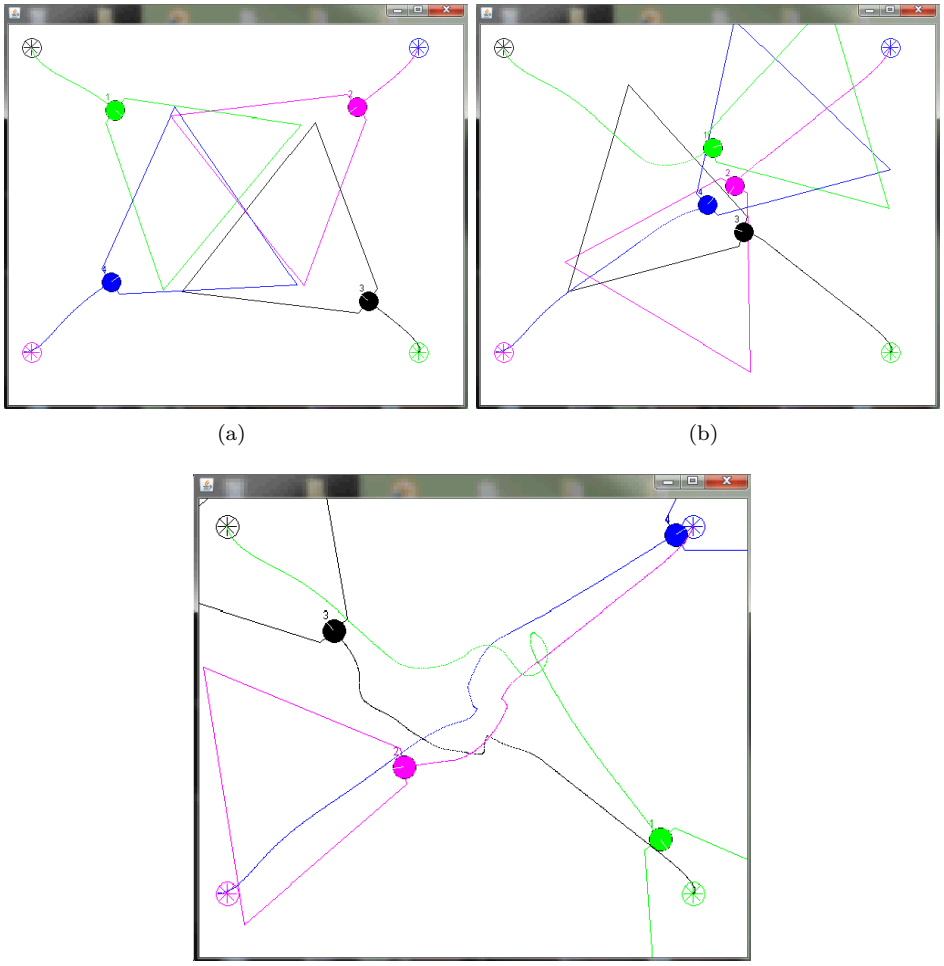
También se modeló el área de detección de los robots de una manera diferente a aplicaciones anteriores. Para este trabajo los robots eran capaces de detectar el área comprendida en un trapecio situado justo delante del robot, emulando las prestaciones reales que más tarde se obtendrían con los robots.

Por simplicidad, para la estrategia de toma de decisiones se consideró en este caso que la carga de desviación fuese siempre compartida en un 50 % entre los robots, es decir, $\alpha = 0,5$.

Se comenzó la experimentación describiendo un escenario sin obstáculos estáticos como el mostrado en la figura 8.9, donde los círculos de colores representan los robots y las circunferencias con 3 diámetros dibujados con su color representan la posición destino que desea alcanzar. Como puede observarse se han definido las posiciones iniciales en las esquinas y las trayectorias de los robots se cruzan en el centro de la arena, lo que produce que en dicho punto se detecten un gran número de colisiones.

La figura 8.9(a) se corresponde con un tiempo de 3 segundos de ejecución cuando los robots todavía no se han detectado entre sí y se encaminan hacia su punto de destino en línea recta. Transcurridos unos segundos, en la figura 8.9(b) puede observarse cómo se producen las primeras detecciones y los resultados de las evasiones mediante la ejecución del algoritmo propuesto. Finalmente, en la figura 8.9(c), se observa cómo cada robot está ya libre de colisión durante toda trayectoria hacia su destino.

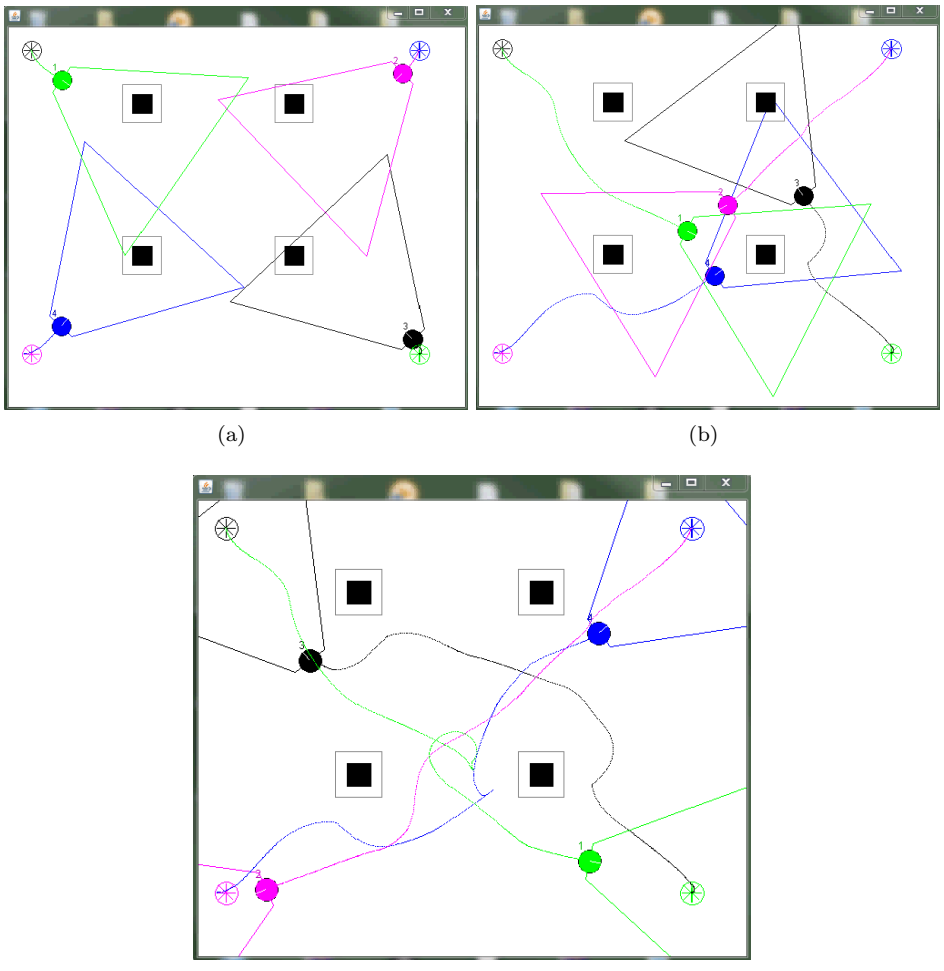
Se generó también un escenario con objetos estáticos representados como cuadros negros en la figura 8.10, donde se representa la secuencia ocurrida durante el experimento. En este caso, cuando un robot, detectaba el obstáculo y preguntaba al conjunto de agentes por su ubicación, no recibía respuesta alguna por lo que se encargaba de evadir la colisión el solo con un $\alpha = 1$.



(a) (b)

(c)

Figura 8.9: Secuencia del algoritmo de evasión de colisiones 1vs1.



(a) (b)

(c)

Figura 8.10: Secuencia del algoritmo de evasión de colisiones 1vs1 con obstáculos estáticos.

Además, dicha simulación se trasladó también al programa de simulación 3D V-REP mediante la integración descrita en el apartado 2.4.4. La imagen 8.11 muestra su ejecución con la simulación de robots Mindstorms NXT y con cubos a modo de obstáculos estáticos.

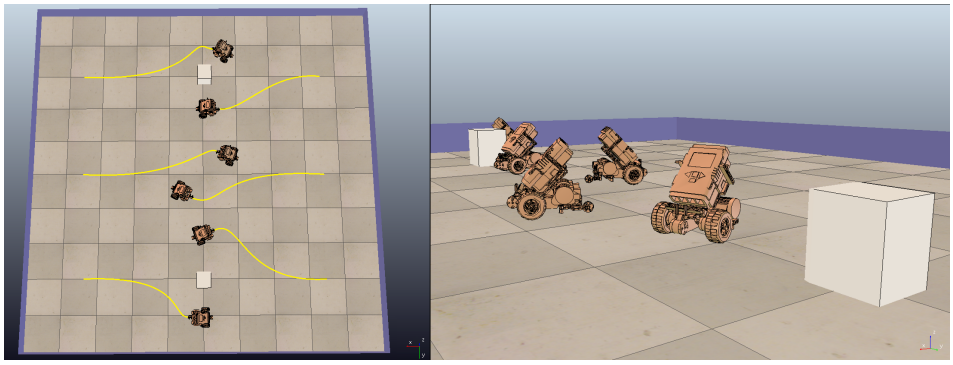


Figura 8.11: Captura de la ejecución del algoritmo de evasión de colisiones en V-REP con LEGO Mindstorms NXT

Cabe destacar que a pesar de utilizar varios robots en la experimentación, el consenso siempre se realiza entre pares de 1 contra 1. Esto provoca que en ocasiones, cuando un robot ya ha alcanzado su posición de evasión, inmediatamente después se vea obligado a evitar a otro robot distinto porque dicha posición genera una nueva colisión. Esta metodología se extrapola para el caso de N contra N en la siguiente sección 8.4.

8.3.0.1 Demostración Experimental con LEGO Mindstorms NXT

Del mismo modo que el resto de algoritmos presentados en la presente tesis, el algoritmo de detección y evasión de colisiones se implementó y se validó en robots móviles de recursos limitados. En este caso en LEGO Mindstorms NXT.

Dado que los sensores del LEGO no tienen un buen alcance, se utilizó la misma metodología que la descrita en la sección 7.2.1 con los e-pucks, para ampliar su rango de detección. La cámara cenital que gobernaba el escenario estaba conectada a un computador que pertenecía al MAS (agente *AMO*) y que era capaz de detectar las posiciones de los robots gracias a la instalación de unos señuelos triangulares sobre las plataformas. La arquitectura del escenario del experimento se muestra en la figura 8.12.

La principal idea del experimento fue replicar las situaciones obtenidas en simulación, pero en este caso ejecutadas sobre dos robots reales. Para que se dieran colisiones

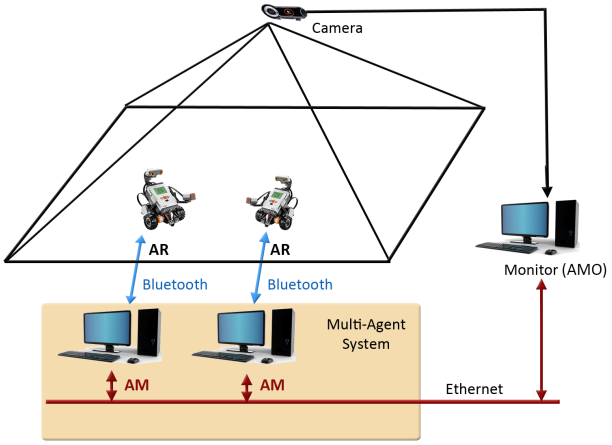


Figura 8.12: Escenario del experimento de evasión de colisiones con dos LEGO Mindstorms NXT.

periódicamente los debían seguir una trayectoria recta definida por un punto inicial y uno final. Una vez se alcanzaba el punto final, se intercambiaban los puntos y los robots daban una vuelta de 180 grados para volver a seguir la trayectoria de vuelta.

El primer problema que surgió con esta experimentación fue que los robots no eran capaces de permanecer fieles a su trayectoria mediante el uso de la odometría después de realizar más de 2 recorridos. Como ocurría en las pruebas experimentales del capítulo 3, mediante el uso de la odometría el error acumulativo provoca que la trayectoria real diverja de la estipulada por el robot. Para corregir este problema hubo que instalar un giróscopo en cada robot e implementar un filtro de Kalman reducido que mejorase la estimación de la postura. Este filtro estaba acoplado dentro del módulo de *Control* del NXT, el cual integraba el seguimiento de trayectoria mediante un algoritmo de persecución pura hacia el punto destino que se le especificaba.

Otro módulo importante era el de *Detección*, mediante el cual el robot era capaz de detectar los obstáculos situados enfrente de él. Tanto el módulo de *Control* como el de *Detección* se comunicaban localmente con el módulo *Comunicación* gracias al cual tenían el enlace con el MAS. La figura 8.13 muestra la relación entre los módulos integrados dentro del NXT.

El funcionamiento del algoritmo era exactamente igual que en simulación. Cuando se detectaba un obstáculo, el agente software representante del robot *detector* preguntaba al MAS si había algún robot en esa posición. En caso afirmativo los dos

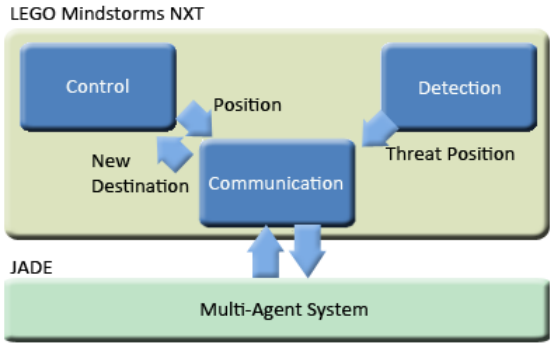


Figura 8.13: Módulos de ejecución en LEGO Mindstorms NXT para el algoritmo de evasión de colisiones.

robots consensuaban una posición de evasión segura y mandaban dicha posición a través del canal de comunicaciones hasta llegar al algoritmo de control, el cual modificaba la posición destino de cada uno. De nuevo, se consideró que los dos robots debían asumir la misma carga de evasión (50 % cada uno).

En la imagen 8.14 se observa una captura del experimento en el instante en el que se asignan nuevas posiciones destino a los robots.

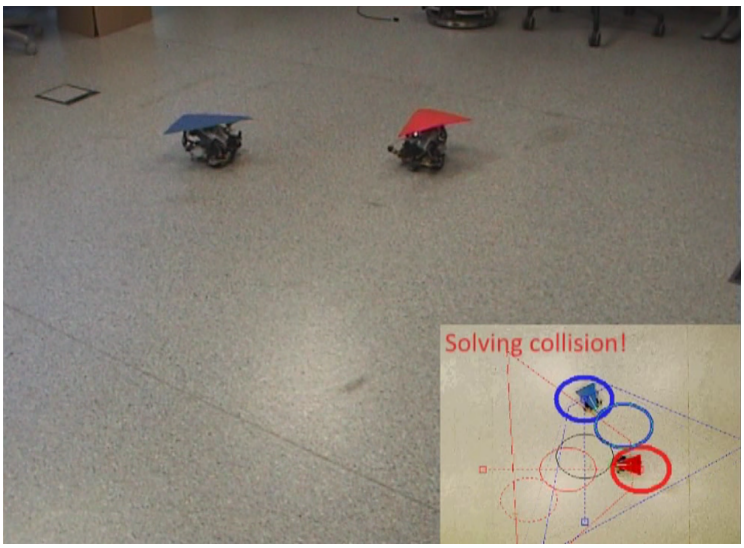


Figura 8.14: Captura del experimento de evasión de colisiones implementado con LEGO Mindstorms NXT.

El vídeo completo del experimento real y en simulación se muestra en la dirección `/https://www.youtube.com/watch?v=V7bZWcepk2c`.

El algoritmo ofrece buenos resultados y el tratamiento de las comunicaciones y los recursos disponibles se gestiona de manera eficiente ya que se ha podido integrar en los robots de recursos limitados LEGO Mindstorms NXT.

Además, el algoritmo es aplicable a estrategias de simplificación de cálculos mediante descomposición espacial, donde la definición de cada celda vendría definida de manera consensuada a través del valor de t_M y en el perímetro de la celda estarían definidas las posiciones iniciales y finales de los robots. Es decir, la división entre espacios se correspondería a las divisiones entre los instantes mínimos de tiempo entre los que hay que evaluar las posibles colisiones. Esto permitiría evaluar el problema de una manera geométrica y además requeriría resolver las colisiones con consensos de robots N a N, ya que la solución del algoritmo debería resolver dentro de cada celda todas las colisiones para todos los robots implicados en ella.

No es habitual que ocurra una colisión entre más de dos robots en un mismo instante de tiempo, sin embargo es un caso plausible para el cual, el algoritmo de consenso 1 vs 1 no aporta ninguna solución, ya que supone que las colisiones siempre ocurren entre dos robots para un mismo instante t_M. El planteamiento de este caso particular derivó en el trabajo que se expone en la siguiente sección.

8.4 Demostración Experimental: Consenso N vs N

Este trabajo se centra en resolver la problemática de la gestión de evasión de colisiones para un agente AC donde a diferencia del apartado anterior, existen más de 2 robots implicados en una colisión en un mismo instante t_M.

Si al trabajar con dos robots, las soluciones de evasión del algoritmo son infinitas teniendo en cuenta el porcentaje de carga de evasión que los agentes acuerdan asignarse entre sí, cuando se trabaja con N robots, aún son superiores.

Por este motivo, para poder abordar el problema, se asume a priori que todos los robots son colaborativos y se ofrecen a cargar mínimo con el 50 % de la evasión. Esto reduce considerablemente las posibilidades de resolución pero no lo suficiente. La posibilidad de cambiar la velocidad de los robots desencadena también un conjunto de soluciones infinitas, sobre todo si se tiene en cuenta por ejemplo la capacidad de parar o retroceder.

Estas dos premisas cambian el enfoque del problema ya que asumiendo que los robots se mantienen a velocidad constante durante toda su trayectoria, para el tiempo t_M es posible definir el conjunto de soluciones como una circunferencia C_i

de posibles posiciones definidas por la ecuación 8.11 de radio $v_i \cdot (t_M - t_s)$ centrada en la posición inicial de cada robot.

$$v_i \cdot (t_M - t_s) = (x - P_i(ts).x)^2 + (y - P_i(ts).y)^2, \forall x, y \in C_i \tag{8.11}$$

La imagen 8.15 muestra parte de dicha circunferencia sobre los robots 1 y 2. Las soluciones deben por tanto estar incluídas en el perímetro de dichas circunferencias.

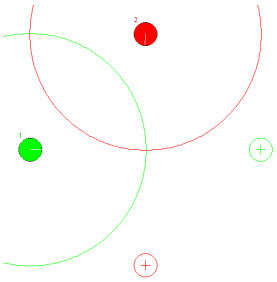


Figura 8.15: Especificación de la circunferencia C_i para los robots 1 y 2.

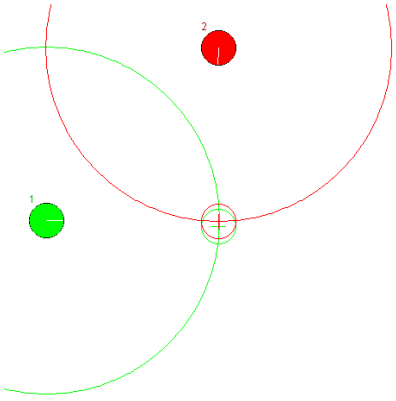
Con este enfoque, la solución óptima al problema con N robots, se puede definir como la configuración que minimiza el sumatorio del ángulo de desviación (θ_i'') de todos los robots tal y como muestra la ecuación 8.12. Siendo θ_i' el nuevo ángulo que debe tener cada robot.

$$min \sum_{i=1}^{N} \triangle\theta_i''(t_s) = min \sum_{i=1}^{N} (\theta_i'(t_s) - \theta_i(t_s)) \tag{8.12}$$

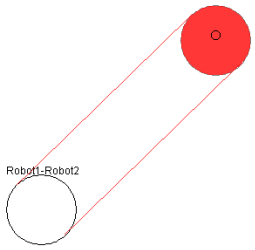
Una de las nuevas modificaciones que hay que aplicar al algoritmo es que las posiciones de evasión $P_i(t_M)$ pertenezcan a C_i definida en 8.11. Esto no se cumple para las posiciones calculadas a partir de las ecuaciones 8.8 y 8.9 ya que el vector $\hat{v}_{MTD}$ se corresponde con una recta en el espacio cartesiano que no puede intersectar con C_i en más de dos puntos.

Se plantea pues la obtención de la proyección de $P_i(t_M)$ sobre C_i y $P_i ts$ de manera que se obtenga el incremento del ángulo correspondiente para encarar dicha posición, sin embargo no sería una buena aproximación ya que el algoritmo calcula $P_i t_M$ para ser alcanzado en el instante t_M. Si la distancia de $P_i t_s$ a $P_i t_M$ es mayor que R_i no se está cumpliendo la condición de que el robot deba moverse a velocidad constante. Si por otro lado, se propone la proyección de $P_i(t_M)$ sobre C_i, el robot debería alcanzar dicha posición en un instante menor que t_M y para ello debería modificar su velocidad, incumpliendo de nuevo que sea constante. Es por este motivo por lo que las evasiones en este trabajo se calculan de manera diferente.

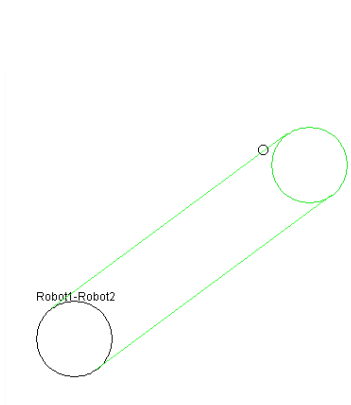
Partiendo de una situación con peligro de colisión entre dos robots, se obtiene en el espacio de Minkowski un segmento el cual representa la diferencia de las trayectorias del espacio euclídeo. Como ya se demostró en la sección 8.2.3, en caso de que ese segmento, cuyo grosor se corresponde con la suma de los radios de los modelos de los robots, pase por el origen de coordenadas del espacio de Minkowski siempre se producirá una colisión entre las trayectorias de los robots. La nueva propuesta de resolución de colisiones se centra en modificar el ángulo del segmento de Minkowski para evitar de ese modo que pase por el origen. Este efecto se traslada al espacio euclídeo como un cambio en la orientación de los robots, la cual habrá que mantener hasta t_M para asegurar la no colisión.



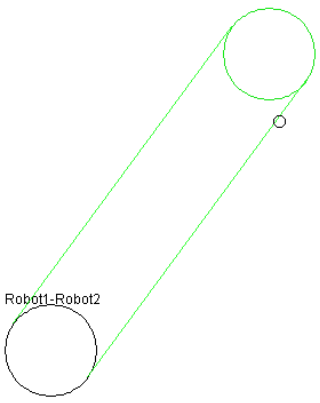
(a) Situación en el espacio euclídeo



(b) Representación en el espacio de Minkowski



(c) Una solución a la colisión en el espacio de Minkowski



(d) Otra solución a la colisión en el espacio de Minkowski

Figura 8.16: Análisis de posibles soluciones de amenaza de colisión entre los robots 1 y 2.

Dado el conjunto de las figuras 8.16, la imagen 8.16(a) muestra el espacio euclídeo donde los robots R_i y R_j presentan una colisión entre sus trayectorias en la posición marcada con su color. La imagen 8.16(b) representa la misma situación en el espacio de Minkowski donde el círculo sin color corresponde a $P_i(t_s) - P_j(t_s)$ y el círculo coloreado a $P_i(t_g) - P_j(t_g)$. Se puede observar que efectivamente existirá colisión entre los robots y por ello se representa de color rojo en la interfaz. Modificando el ángulo del segmento en el espacio de Minkowski es posible evitar que dicho segmento cruce el origen de coordenadas. Existen dos soluciones posibles para ello: la mostrada por la figura 8.16(c) y la representada en la figura 8.16(d).

De las dos soluciones, la óptima es sin duda la que supone una menor variación del ángulo de los dos robots. Pero si se trabaja con colisiones entre más de 2 robots, pueden ocurrir casos en los que la elección de una evasión mínima por parte de un robot, provoque la evasión máxima por parte de otro, por lo que se contemplan ambos casos y se ramifican en las posibles soluciones hasta encontrar el mínimo.

Los dos ángulos δ y γ que definen las dos variaciones del ángulo se detallan en la figura 8.17. δ para evadir hacia el lado derecho y γ para evadir por el izquierdo. Un cambio de orientación en Minkowski en base a cualquiera de los dos ángulos, haría que el segmento ya no incluyese el origen de Minkowski, por lo que solucionaría la colisión.

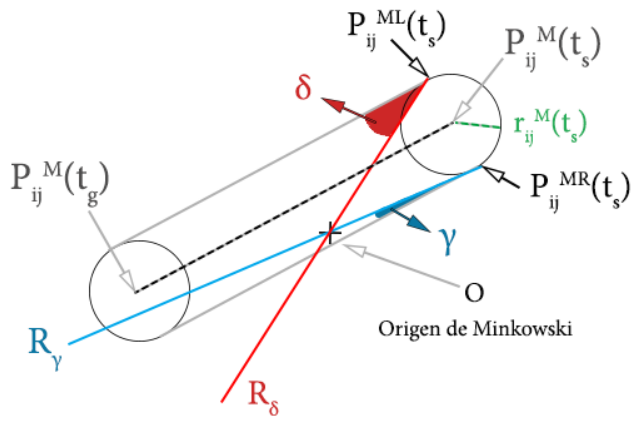


Figura 8.17: Especificación de los ángulos δ y γ en el espacio de Minkowski.

Para realizar el cálculo de estos ángulos primero se calculan los puntos de los extremos $P_{ij}^{ML}(t_s)$ (izquierdo) y $P_{ij}^{MR}(t_s)$ (derecho) en la posición origen en Minkowski $P_{ij}^{M}(t_s)$ gracias a las ecuaciones 8.13.

$$\begin{aligned} P_{ij}^{ML}(t_s) &= (P_{ij}^{M}(t_s).x + (R_{i,j}^{M} \cdot \cos(\theta_{i,j}^{M})), (P_{ij}^{M}(t_s).y - (R_{i,j}^{M} \cdot \sin(\theta_{i,j}^{M}))) \\ P_{ij}^{MR}(t_s) &= (P_{ij}^{M}(t_s).x - (R_{i,j}^{M} \cdot \cos(\theta_{i,j}^{M})), (P_{ij}^{M}(t_s).y + (R_{i,j}^{M} \cdot \sin(\theta_{i,j}^{M}))) \end{aligned} \tag{8.13}$$

Proyectando las rectas R_δ y R_γ sobre la recta que define la orientación del segmento de Minkowski, se pueden obtener las posiciones de intersección P_{delta} y P_γ de las cuales se pueden calcular ángulos equivalentes a δ' y γ' tal y como muestra la figura 8.18.

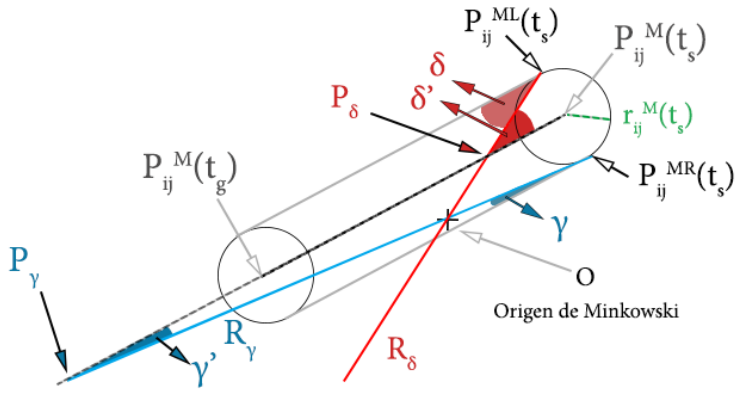


Figura 8.18: Especificación de los ángulos δ' y γ' en el espacio de Minkowski.

Dichos ángulos se pueden calcular por trigonometría mediante las ecuaciones 8.14 donde $r_{ij}^{M}(t_s)$ se corresponde con el radio de la posición inicial en Minkowski.

$$\begin{aligned} \delta &= \arccos\left(\frac{r_{ij}^{M}(t_s)}{\sqrt{\left(P_\delta.x - P_{ij}^{MR}(t_s).x\right)^2 + \left(P_\delta.y - P_{ij}^{MR}(t_s).y\right)^2}}\right) \\ \gamma &= \arccos\left(\frac{r_{ij}^{M}(t_s)}{\sqrt{\left(P_\gamma.x - P_{ij}^{ML}(t_s).x\right)^2 + \left(P_\gamma.y - P_{ij}^{ML}(t_s).y\right)^2}}\right) \end{aligned} \tag{8.14}$$

El mínimo de ambos ángulos (en el ejemplo de la figura, *gamma*) será el cual genere la menor variación de ángulo en el espacio euclídeo. Por lo que el algoritmo calcula ambos ángulos para cualquier colisión detectada entre pares de robots y para ello es necesario calcular todas las posibles colisiones que pueden ocurrir cuando se trabajan con N robots.

El número máximo de colisiones se puede definir según la teoría de la combinatoria como el número de combinaciones de N elementos diferentes tomados en este caso de 2 en 2 ($k = 2$) porque a pesar de que en una misma colisión participen más de dos robots, las colisiones se tratan por cada pareja de robots. La ecuación 8.15 muestra la fórmula general para cualquier caso.

$$C_k^N = \frac{N!}{k!(N-k)!}, k \leq N. \tag{8.15}$$

Se calcula el número total de colisiones Nc en el instante t_M donde cada colisión C_{ij} viene definida por el par de robots que intervienen $< R_i, R_j >$ y sus posiciones para t_s y t_M. Para cada colisión entre pares (abstraída a partir de ahora como C_u, con $u \in [0, Nc-1]$), existen dos soluciones posibles (δ o γ) por lo que es posible construir un árbol de soluciones ramificando en función de dos factores: la colisión objetivo seleccionada a resolver cada iteración y la solución aplicada en dicha colisión.

Es importante el orden en el que se van seleccionando las colisiones a resolver ya que resolver una colisión genera una situación diferente sobre la que se resuelven las demás. Por este motivo se ramifica tanto en función de la solución escogida como del orden de la selección de las colisiones.

La figura 8.19 muestra el esquema general del árbol generado. El estado del nodo raíz está formado por el conjunto de colisiones existentes en t_M en la situación inicial $< C_0, ...C_{Nc} >$ y tiene un índice de coste $< 0 >$. El algoritmo comienza escogiendo una colisión C_i y calculando los valores de C_i^δ y C_i^γ que la resuelven. Si aplicando las soluciones se genera una situación en la que no se genera una nueva colisión, se crea un nodo nuevo para cada una, situado a un nivel inferior con un estado definido por el número de colisiones pendientes por resolver (como máximo $Nc-1$) y un índice de coste igual al coste del nodo padre más el ángulo aplicado para resolver la última colisión (C_i^δ o C_i^γ). Además, se marcan los dos robots implicados en C_i como *visitados* para no volver a modificar su ángulo en dicha rama evitando así la posibilidad de bucles.

La construcción del árbol prioriza la expansión de los nodos cuyo índice de coste es menor. De este modo cuando se alcanza un nodo *hoja* donde $N_c = 0$ y su índice se corresponde con el menor del conjunto de posibles nodos *solución*, se alcanza la solución óptima. El camino seguido desde la raíz hasta el nodo *hoja-solución* define

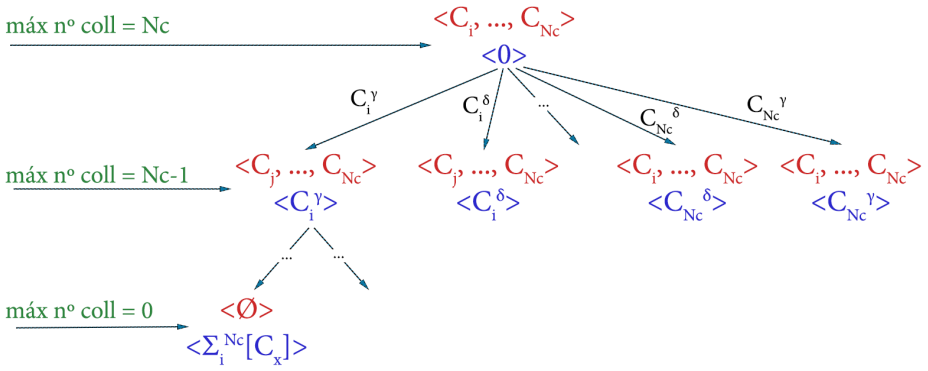


Figura 8.19: Árbol de soluciones generado para resolver las colisiones entre N robots.

la secuencia de ángulos a aplicar para resolver cada colisión y el coste de dicho nodo se corresponde con la suma de las variaciones de ángulos que han generado dicha solución.

El máximo de nodos que se pueden expandir desde cualquier nodo con Nc colisiones es de $2 \cdot Nc$ y la máxima profundidad que puede alcanzar el árbol es de Nc contando el nodo raíz como profundidad 0, por lo que el número máximo de nodos creados viene definido por la ecuación 8.16.

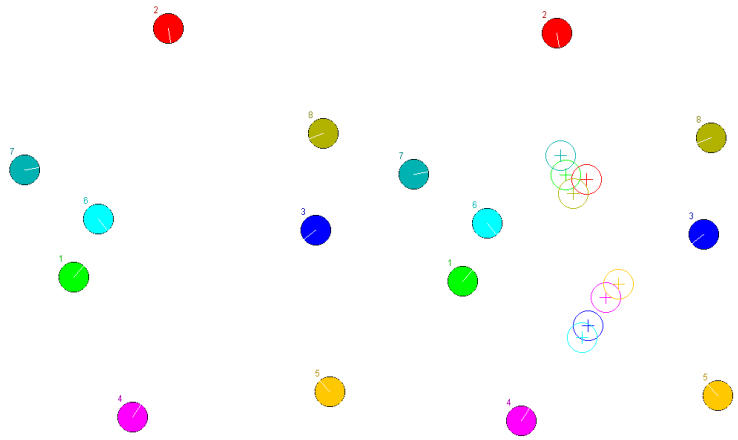
$$1 + \sum_{i=1}^{Nc} \left(\prod_{u=i}^{Nc} (u \cdot 2) \right) \tag{8.16}$$

Naturalmente, puede ocurrir que al resolver una colisión (generar un nuevo nodo), dicha solución resuelva más de una colisión y por tanto algún nodo *hoja* quede en una profundidad inferior a Nc por lo que el árbol generado no tiene por qué estar perfectamente equilibrado. Esto ocurre por ejemplo, cuando un robot R_i colisiona con los robots R_j y R_k y una solución a la colisión C_{ij} resuelve a su vez la colisión C_{ik}.

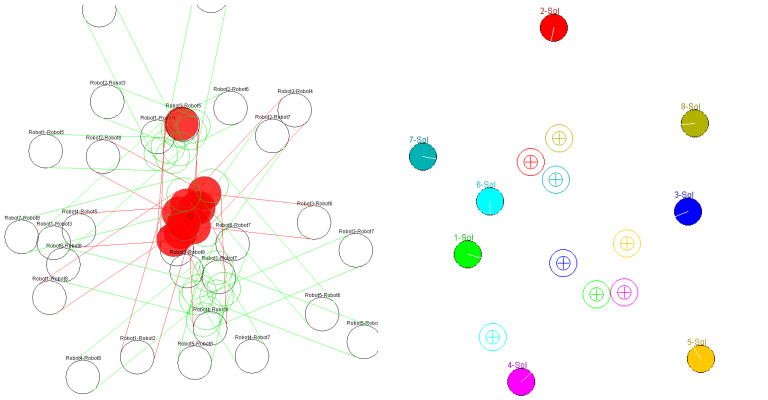
El algoritmo 17 muestra el pseudocódigo de la metodología implementada en el agente AC encargado de resolver la colisión N vs N.

Algoritmo 17: Pseudocódigo del algoritmo de evasión de colisiones propuesto para consensos de N vs N

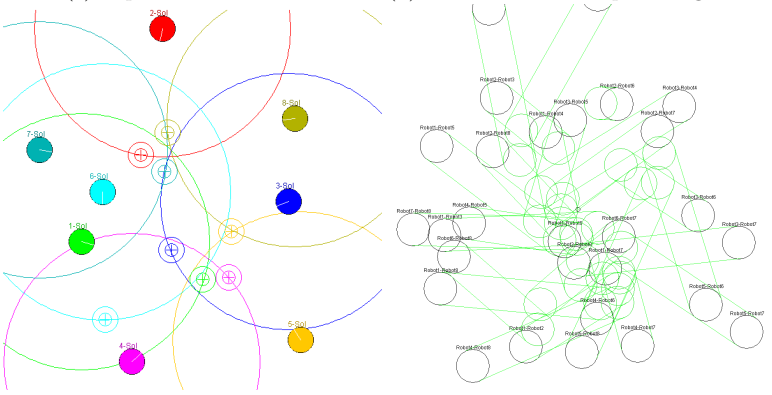
Entrada:
<$P_i(t_s)$, $P_i(t_g^i)$, r_i, v_i>$\cdots$<$P_N(t_s)$, $P_N(t_g^N)$, r_N, v_N>: *Parámetros de* $R_i, \cdots, R_N$
Salida:
Solucion =< $\theta_i', \cdots, \theta_N'$ > *donde,*
<θ_i'>, $\cdots$, <θ_N'>: *Nuevos ángulos de* $R_i, \cdots, R_N$
Inicialización:
Estados=[]
//Cálculo de todas las colisiones entre pares de robots
Para *cada Robot* R_i *de* G_R *hasta* R_N **hacer**
  Para *cada Robot* R_j *con* $i! = j$ *de* G_R *hasta* R_N **hacer**
    Si ∃*Collision(*R_i,R_j*)* **entonces**
      E.Colisiones← *Add(*$C_{ij}(R_i, R_j)$*)*
      E.valor=0
    fin
  fin
fin
//Creación y exploración del árbol
Para *cada* $E \in$ *Estados* **hacer**
  //Nodo *hoja-solución*
  Si *E.Colisiones.size=0 && E.valor=min(*∀ *Estados.valor)* **entonces**
    Solucion=E
    return Solucion
  si no
    //Extraer el estado de mínimo valor del conjunto de estados y ramificarlo
    E ← Extraer_Estado (min(∀ Estados.valor))
    Expandir(E)
  fin
fin
//método de expansión de nodos.
Expandir:
Para *cada* $C_u \in$ *E.Colisiones* **hacer**
  $[E_\delta, E_\gamma]$ ← *Evasión_colisión(*C_u*)*
  Si E_δ*.Colisiones.size<E.Colisiones.size* **entonces**
    Estados.Add(E_δ*)*
  fin
  Si E_γ*.Colisiones.size<E.Colisiones.size* **entonces**
    Estados.Add(E_γ*)*
  fin
fin



(a) Situación inicial, espacio euclídeo (b) Posiciones de máxima aproximación



(c) Espacio de Minkowski (d) Solución calculada por el algoritmo



(e) Soluciones ∈ ecuación 8.11 (f) Solución en el espacio de Minkowski

Figura 8.20: Ejemplo de solución de una situación con 8 robots

La figura 8.20 muestra la secuencia de un ejemplo de ejecución del algoritmo presentado en un escenario con 8 robots.

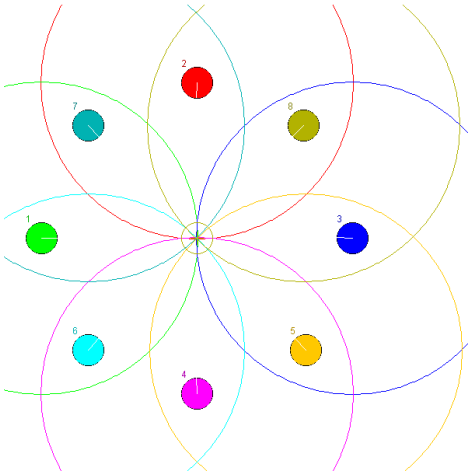
La imagen 8.20(a) recoge la situación inicial donde resulta difícil intuir qué colisiones ocurrirán en ejecución y entre qué robots. Una vez calculado el instante máximo de aproximación, se obtienen las posiciones mostradas en la figura 8.20(b) las cuales revelan tres grupos diferenciados de colisiones. En concreto, el escenario de Minkowski de la figura 8.20(c) revela que habrá un total de 9 colisiones divididas entre tres grupos de robots aparentemente. Eso quiere decir que se crean tres agentes AC en total, uno para cada grupo de colisiones. Ejecutando el algoritmo de evasión de colisiones sobre cada uno, se obtienen las nuevas posiciones libres de colisión mostradas en la figura 8.20(d), las cuales cumplen la condición de la ecuación 8.11 tal y como muestra la figura 8.20(e). Su correspondiente representación en Minkowski (figura 8.20(f)) verifica que no existirá colisión entre ellas por lo que el algoritmo ha resuelto las 9 colisiones correctamente.

Resulta complicado en situaciones como la de la figura 8.20 observar la optimalidad de las soluciones del algoritmo propuesto. Por este motivo y para comprobar la robustez del algoritmo, se plantean unos escenario *límite*, donde el grupo de robots que están presentes en el escenario colisionan todos al mismo tiempo y en el mismo punto. Esto quiere decir que únicamente un agente AC es el encargado de gestionar todas las colisiones de N agentes contra N agentes.

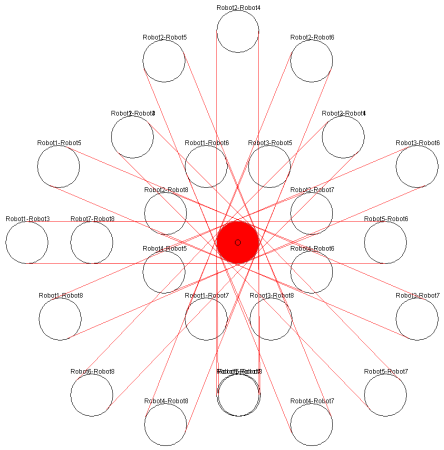
Las figuras 8.21 y 8.22 muestran un escenario *límite* con 8 y 16 robots respectivamente, los cuales todos colisionan en el mismo punto y en el mismo instante de tiempo, es decir, que el número de colisiones detectadas en cada caso se corresponde con el máximo, $C_2^8 = 28$ para 8 robots y $C_2^16 = 120$ para 16 robots.

En las imágenes 8.21(a) y 8.22(a) se expone la situación inicial en el espacio euclídeo para los dos casos y las imágenes 8.21(b) y 8.22(b) describen la situación en el escenario de Minkowski donde se observan múltiples colisiones entre todos los robots en ambos casos. Las figuras 8.21(c) y 8.22(c) muestran el resultado devuelto por el AC tras ejecutar el algoritmo. Las nuevas posiciones de evasión en Minkowski ya no generan ninguna colisión como se muestra en las figuras 8.21(d) y 8.22(d) y los robots pueden seguir la nueva trayectoria sin colisionar entre ellos.

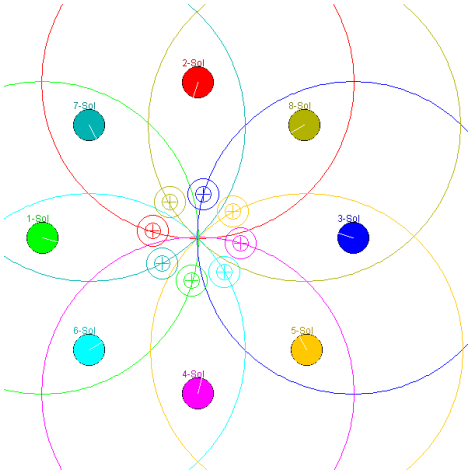
Estas situaciones *límite* ofrecen un resultado óptimo ya que la posición de evasión de la colisión no puede estar más ajustada para que no exista colisión entre ningún robot. Esto verifica la optimalidad del algoritmo y su capacidad de ofrecer soluciones en las situaciones más complejas.



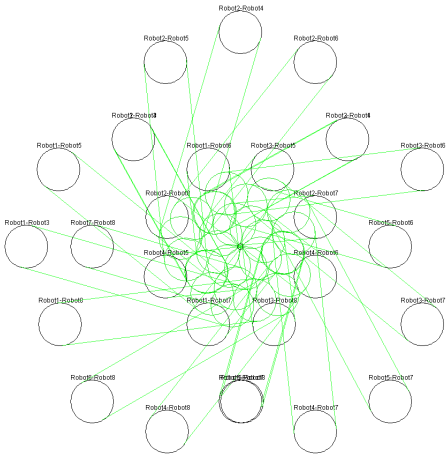
(a) Situación en el espacio euclídeo



(b) Representación en el espacio de Minkowski

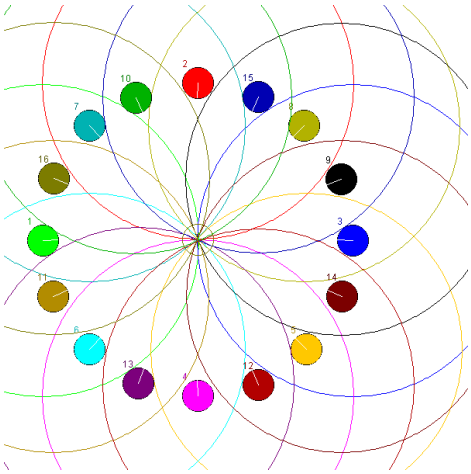


(c) Solución calculada por el algoritmo de consenso N vs N

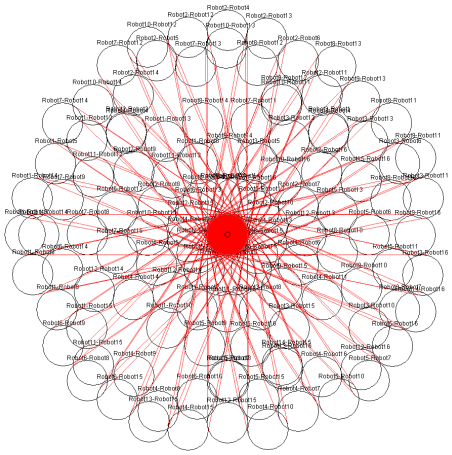


(d) Representación de la solución a las colisiones en el espacio de Minkowski

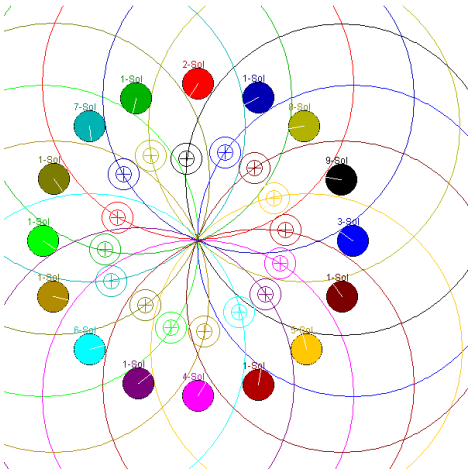
Figura 8.21: Ejemplo de solución de una situación límite con 8 robots



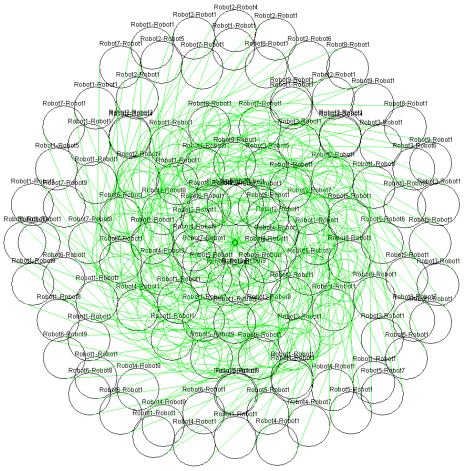
(a) Situación en el espacio euclídeo



(b) Representación en el espacio de Minkowski



(c) Solución calculada por el algoritmo de consenso N vs N



(d) Representación de la solución a las colisiones en el espacio de Minkowski

Figura 8.22: Ejemplo de solución de una situación límite con 16 robots

La tabla 8.1 refleja diversas pruebas de ejecución del algoritmo en la situación *límite* cuando todos los robots colisionan con todos en la misma posición y el mismo instante.

Nº de robots	Nº de colisiones	Nodos creados	T. de solución(ms)
3	3	79	30
4	6	633	75
6	15	75973	103
8	28	17017969	195
10	45	$6.1265 \cdot 10^{9}$	340
12	66	$3.2348 \cdot 10^{12}$	505
14	91	$2.3549 \cdot 10^{15}$	811
16	120	$2.2607 \cdot 10^{18}$	1200

Tabla 8.1: Resultados de las pruebas con el algoritmo N vs N en situaciones *límite*

Los tiempos de ejecución se tomaron sobre un ordenador Intel(R) Core(TM)2 Quad Q9550 a 2.83GHz y se calcularon como la media de 10 ejecuciones. Al graficar la relación entre los distintos elementos que intervienen en el algoritmo se observa que su coste asintótico es exponencial en cuanto al número de robots implicados, tal y como muestra la gráfica 8.23. Es natural, ya que el número de colisiones y, por consiguiente, el número de nodos generado también lo son.

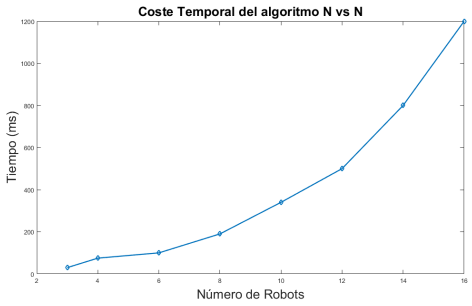


Figura 8.23: Gráfico del coste temporal del algoritmo en las distintas pruebas.

Por último, cabe destacar que de igual modo que el algoritmo 1 vs 1, el marco del problema N vs N también es aplicable a estrategias de descomposición espacial donde cada celda vendría definida por una posición de entrada y una posición de salida de los robots. Las distintas colisiones ocurridas dentro de cada celda generarían una posición intermedia que evadiría la colisión y aseguraría que el robot sale de la celda por la misma posición final que tenía asignada originalmente.

En la dirección `/https://www.youtube.com/watch?v=eBteHxgxdwU` se muestra un vídeo de diversas ejecuciones del algoritmo propuesto, resolviendo distintas situaciones.

8.5 Conclusiones del Capítulo

En el presente capítulo se propusieron y desarrollaron diversos algoritmos relacionados con la detección y evasión de colisiones de robots móviles con recursos limitados, haciendo uso de metodologías basadas en agentes.

- Se ha detallado las metodologías de evasión de colisiones más extendidas en el campo de la robótica y descrito el marco dentro el cual se define un nuevo algoritmo propuesto de detección de colisiones mediante el consenso de agentes 1 vs 1 basado en una arquitectura distribuida.
- Se ha definido el nuevo algoritmo de detección y evasión de colisiones así como las fases que lo componen y sus requerimientos.
- Se ha implementado el algoritmo aportado de detección y evasión de colisiones mediante consenso 1 vs 1 sobre una plataforma simulada, validando su ejecución y desarrollo al ofrecer resultados óptimos de resolución de colisiones.
- Se ha aplicado el algoritmo anterior, junto con la arquitectura basada en el modelo de agentes a la plataforma de recursos limitados LEGO Mindstorms NXT, obteniendo resultados satisfactorios durante todas las pruebas experimentales.
- Se ha extendido el algoritmo a un contexto de consenso entre robots de N vs N, donde las colisiones se resuelven de manera global para un grupo de robots implicado en una colisión.
- El algoritmo aportado de detección y evasión de colisiones mediante consenso N vs N se ha integrado en la plataforma de simulación y se ha reproducido situaciones límite complejas con un gran número de robots para comprobar la optimalidad y convergencia del algoritmo, ofreciendo en todos los casos resultados óptimos.

Capítulo 9

Conclusiones y Trabajos Futuros

Este capítulo recoge las conclusiones obtenidas a partir del desarrollo de la presente tesis así como las futuras líneas de investigación sobre los métodos propuestos para llevar a cabo en trabajos futuros. Al final del capítulo se exponen todos los artículos presentados en revistas, capítulos de libro y congresos, derivados de los resultados obtenidos en este trabajo.

9.1 Conclusiones

Teniendo en cuenta los antecedentes expuestos en cada capítulo y las distintas arquitecturas descritas en el capítulo 2, se determinó el beneficio del uso de sistemas distribuidos para lidiar con las problemáticas asociadas a la robótica móvil. Estos sistemas ofrecen flexibilidad, escalabilidad, modularidad y son fácilmente escalables, reproducibles y reconfigurables. Además, los sistemas multi-agente ofrecen la posibilidad de dotar a los agentes robóticos de la capacidad de comunicarse entre sí, brindando la opción de poder conversar, negociar y consensuar decisiones críticas. El desarrollo de este marco de arquitectura permite además aplicar posibles estrategias del campo de la *IA* ya utilizadas en otros ámbitos.

Sobre este contexto se han desarrollado algoritmos para resolver tareas importantes como la localización local y global de robots en interiores y exteriores, la navegación y construcción de mapas del entorno, la coordinación entre grupos de robots para conseguir un beneficio individual que mejorase a su vez el colectivo, y la detección y evasión de colisiones basada en consensos entre robots. Siempre con la finalidad de ofrecer una mayor autonomía a los robots tanto en tomas de decisiones a nivel individual como a nivel colectivo.

Todos los algoritmos desarrollados se han aplicado tanto en tareas de simulación como en experimentos prácticos implementados sobre robots reales, obteniendo resultados satisfactorios pese a la problemática añadida que ello conlleva. Además, la resolución a las problemáticas se ha resuelto desde un punto de vista práctico y orientado a su aplicabilidad en plataformas de recursos limitados en capacidad de cómputo, memoria, prestaciones y comunicaciones.

En la presente tesis se ha cumplido con los siguientes objetivos específicos:

- Se realizó una revisión del estado del arte de RSFs orientados a sistemas distribuidos, así como de las tareas de localización local, global y cooperativa de robots mediante filtros de fusión sensorial, de las tareas de navegación basadas en procesos independientes y de las metodologías de detección y evasión de colisiones.
- Se investigó sobre los distintos RSFs aplicables a los robots utilizados en las pruebas experimentales así como su metodología de programación, el funcionamiento de sus sensores y la gestión de sus comunicaciones.
- Se aportó a la plataforma JADE un nuevo agente supervisor denominado *debugger* encargado de realizar tareas de monitorización, interacción entre robots y registro de mensajes enviados y recibidos. El cual se utilizó para corrección e identificación de errores durante el desarrollo de los algoritmos de este trabajo.
- Se desarrolló el middleware JADE-LeJOS orientado al uso de la plataforma multi-agente JADE junto con el robot LEGO Mindstorms NXT gestionado con el firmware LeJOS.
- Se ha presentado el middleware de control y comunicación $J-R$ desarrollado, capaz de integrarse en sistemas robóticos que trabajan bajo la extendida arquitectura distribuida ROS; pudiendo así formar parte de un sistema multiagente basado en JADE que hace uso de las ventajas de los protocolos de comunicación basados en la especificación FIPA-ACL.
- Se realizó la integración de JADE con V-REP para poder visualizar demostraciones a nivel de simulaciónemulación con los distintos modelos de robots que incorpora V-REP.
- Para trabajar en escenarios complejos donde la aplicabilidad de los algoritmos desarrollados no era posible de migrar a plataformas experimentales reales, bien por falta de medios o de logística, se desarrolló una plataforma de emulación de robots 2D basada en el lenguaje de programación JAVA.
- Se experimentó con las distintas plataformas utilizadas para las demostraciones experimentales de la presente tesis, analizando y aprendiendo sus distintos

RSFs e investigando sobre los límites alcanzados por sus prestaciones. Además se realizaron las tareas pertinentes de calibración de sensores así como el estudio de la gestión de los protocolos de comunicaciones que cada robot ofrece para conocer a fondo las capacidades de cada uno de ellos.

- Se obtuvieron modelos dinámicos locales y cinemáticos globales de un robot móvil en configuración diferencial y Ackerman, utilizando métodos de descomposición por partículas inerciales. Con estos modelos se obtiene la dinámica de la plataforma al colocar acelerómetros en posiciones adecuadas, reduciendo la cantidad de parámetros que se deben identificar, así como la complejidad computacional de los mismos. Se identificaron además los modelos dinámicos de velocidad y aceleración para los motores de corriente continua de la plataforma LEGO NXT empleada en las pruebas experimentales.

- Se presentó el algoritmo desarrollado de localización global geométrica $GEMA^2$ orientado a la localización global de robots de recursos limitados con un láser range finder embarcado. Se aplicó el algoritmo al robot omnidireccional Robotino y se expusieron los resultados favorables de las pruebas de experimentación en cuanto a fiabilidad, flexibilidad y robustez del algoritmo.

- Se realizó un estudio sobre los métodos actuales en estimación de estados y fusión sensorial, a partir del cual se seleccionaron las técnicas de fusión basadas en el filtro de Kalman.

- Se seleccionaron las estrategias de fusión sensorial basadas en los filtros EKF, UKF y KF; y se realizaron diversas reducciones temporales y computacionales consiguiendo su integración completa en plataformas de recursos limitados.

- Se desarrollaron filtros de fusión sensorial de correción continua (TCLA) para la mejora de la estimación de la localización local de vehículos, los cuales no consideran los problemas de acceso a la información de posicionamiento global.

- Para mejorar los resultados del algoritmo de corrección continua y extender el método a una localización global, se desarrolló un algoritmo de localización local y global basado en eventos (ECLA).

- Se realizó una definición del evento adecuada al caso de localización de robots móviles, utilizando la covarianza del error de estimación de la postura del robot como el conocimiento que el robot tiene de sí mismo sobre su mal posicionamiento.

- A partir de la consciencia creada en cada agente robótico sobre su mal posicionamiento, se implementó el algoritmo de fusión sensorial por eventos (ECLA) para corregir puntualmente la postura del robot de manera global.

- Mediante el algoritmo desarrollado de actualización global por eventos (ECLA), se logró reducir la utilización del ancho de banda entre el robot y el sensor global de medición, utilizando este sensor únicamente si la consciencia del robot excedía cierto umbral determinado. Se realizaron distintas definiciones del valor evento y de su límite, demostrando su validez para la navegación en interiores y exteriores de robots de recursos limitados.

- Se presentó un esquema eficiente de localización distribuida basada en eventos y con sucesos cooperativos entre agentes robóticos. Adicionalmente a las posibles misiones definidas entre los agentes, se dotó al sistema de la capacidad de cooperar para conseguir mejorar la localización de un conjunto de robots heterogéneos a partir de las mediciones de las distancias relativas entre agentes.

- A partir de las pruebas experimentales desarrolladas para las distintas plataformas, se realizó la comparativa entre el método de localización cooperativa temporal (TCLA) y el método por eventos (ECLA), demostrando empíricamente las ventajas del ECLA en cuanto a la reducción del uso del canal de comunicación y la mejora de la precisión de la localización, consiguiendo una solución de compromiso entre estos dos factores. Se observó un desempeño adecuado y un error acotado regulado por el nivel del evento de los algoritmos propuestos.

- Se ha aportado un algoritmo de navegación autónoma de vehículos basado en una arquitectura distribuida donde cada nodo o módulo de ejecución se encarga de gestionar procesos independientes entre sí, los cuales mediante una comunicación basada en publicación-suscripción consiguen recopilar información para ejecutar módulos más complejos.

- Se ha descrito un algoritmo de SLAM desarrollado mediante un filtro de partículas basado en la arquitectura ROS e implementado sobre el módulo *Mapping*.

- A partir del mapa generado por el módulo *Mapping*, se ha presentado un algoritmo de localización basado en Montecarlo y en filtro de partículas y correspondencias entre las medidas de un láser embarcado, la odometría y el mapa conocido.

- Se mostró una demostración experimental de ambos algoritmos ejecutados en distintas plataformas con resultados satisfactorios incluso cuando intervienen outliers durante su ejecución.

- Se ha propuesto una metodología apropiada para dotar de mayor y mejor sensorización a robots móviles de recursos limitados mediante el uso de MAS.

- Se presentó la arquitectura definida para el modelo de agentes utilizado para desarrollar un algoritmo de agrupaciones holónicas mediante la coordinación y navegación de robots móviles basado en arquitecturas distribuidas MAS.
- Se implementó dicha arquitectura primero en simulación y posteriormente en robots móviles de recursos limitados ofreciendo buenos resultados y consolidando la buena ejecución del algoritmo de coordinación.
- Se analizó las distintas metodologías de evasión de colisiones más extendidas en el campo de la robótica y se describió el marco dentro el cual se definió el nuevo algoritmo propuesto de detección de colisiones mediante el consenso entre agentes 1 vs 1 basado en una arquitectura distribuida.
- Se implementó el algoritmo aportado de detección y evasión de colisiones mediante consenso 1 vs 1 sobre una plataforma simulada, validando su ejecución y desarrollo y posteriormente se implementó en el robot LEGO Mindstorms NXT, obteniendo resultados satisfactorios durante todas las pruebas experimentales.
- Se extendió el método del algoritmo para llevar a cabo el consenso entre robots de N vs N, donde las colisiones se resuelven de manera global para un grupo N de robots implicado en una colisión.
- El algoritmo de evasión de colisiones por consenso N vs N se integró en la plataforma de simulación donde se reprodujeron situaciones límite complejas con un gran número de robots para comprobar y verificar la optimalidad y convergencia del algoritmo, ofreciendo en todos los casos resultados óptimos.

9.2 Trabajo Futuro

Partiendo de los resultados obtenidos en la presente tesis, a continuación se exponen distintas líneas de investigación sobre las que trabajar en un futuro:

- Investigar sobre cómo llevar a cabo la integración completa de las capacidades de la plataforma JADE en el RSF ROS sin la necesidad de ejecutar una máquina virtual JAVA, para reducir así el consumo de recursos cuando se utiliza el middleware $J-R$ desarrollado en el presente trabajo. Una migración del código de JADE al lenguaje de programación C++ podría integrar sus funcionalidades unificando ambas plataformas en una sola dotada de las posibilidades que las dos ofrecen con el $J-R$.
- Añadir realimentación de los resultados de localización global obtenidos en iteraciones previas al algoritmo $GEMA^2$. Esto aceleraría la convergencia del algoritmo y fortalecería su robustez frente a outliers.

- El algoritmo $GEMA^2$ es extensible a otro tipo de mapas compuestos por otras formas geométricas más complejas. Además de la inferencia en líneas, caracterizando las formas geométricas se podría inferir sobre las mismas, ampliando de este modo su exactitud y su velocidad de convergencia.

- En caso de poder aplicar los algoritmos a robots sin recursos limitados se puede investigar sobre la integración de otras versiones del filtro de Kalman. Una opción interesante es estudiar la viabilidad de incorporar filtros adaptativos con ajuste en tiempo real de sus parámetros (Matrices Q_k,R_k, modelo, etc).

- A los esquemas de fusión propuestos, se pueden incorporar otras representaciones sobre la orientación del robot, como ángulos de Euler, cuaterniones o matrices de rotación, de modo que proporcionen ventajas en la fusión entre los distintos sensores de orientación, a pesar de requerir en algunos casos de transformaciones no lineales.

- En este trabajo no se ha podido aprovechar las prestaciones de un sensor tan importante como es la brújula para el cálculo de la orientación de los robots. Se debe a la sensibilidad de estos sensores por los campos magnéticos del entorno. Estudiar la posibilidad de aislar dichas perturbaciones permitiría dar mayor peso a las medidas de estos sensores y mejorar notablemente la estimación del cálculo de la orientación de los robots.

- Dentro del contexto de las plataformas de recursos limitados se debería estudiar a fondo el rendimiento (carga computacional utilizada/carga computacional total) entre los distintos agentes que forman parte del sistema para tomar decisiones sobre dicho rendimiento. El evento definido en el algoritmo de fusión global por eventos (ECLA) se puede redefinir de manera que refleje un índice en tiempo real de la capacidad de cómputo disponible en cada robot en cada momento o incluso un índice que suponga la ponderación de distintos factores como puede ser el nivel de batería de cada momento, el porcentaje de ocupación de CPU, el retardo en las comunicaciones, las prestaciones de los sensores, etc.

- Las limitaciones de velocidad de los protocolos de comunicaciones utilizados (mayormente Bluetooth 2.0) también han supuesto una fuerte restricción temporal a la hora de plantear posibilidades de resolución de problemas. Poder trabajar con plataformas más modernas equipadas con Bluetooth 4.0 o Wi-Fi abre un abanico de posibilidades en cuanto a la distribución de la carga computacional de los algoritmos y al número de interacciones entre agentes ante una situación crítica en el tiempo.

- El algoritmo de evasión de colisiones 1 vs 1 y N vs N, es aplicable a otros tipos de trayectorias definidas mediante modelos matemáticos como son las curvas de Bezier o las Splines. La aplicación de esta metodología a este tipo

de trayectorias abre un gran abanico de posibilidades dentro del campo de la planificación de movimientos mediante modelos geométricos.

- Todos los algoritmos desarrollados en esta tesis se han aplicado sobre robótica móvil por lo que únicamente se ha tenido en cuenta el espacio de trabajo 2D. Es posible extender las aplicaciones propuestas a otras plataformas de robots como son los equipos híbridos, submarinos o aéreos, en cuyo caso se debe considerar un espacio 3D en lugar del plano 2D y adecuar los algoritmos desarrollados en esta tesis en este sentido.

9.3 Artículos Publicados

La autoría y vigencia de la presente tesis queda validada con la variedad de publicaciones realizadas a partir de los resultados obtenidos en la misma, los cuales cumplen con todos los objetivos propuestos. Estos artículos presentados en libros, revistas y revistas se detallan a continuación.

9.3.1 Capítulos de Libro:

- **A. Soriano**, E. J. Bernabeu, A.Valera, M.Vallés. "Multi-Agent Systems Platform for Mobile Robots Collision Avoidance.". En: *Advances on Practical Applications of Agents and Multi-Agent Systems, Ed. Springer*, ISBN: 978-3-642-38073- 0, pp. 320-324, 2013.

- **A. Soriano**, E. J. Bernabeu, A.Valera, M.Vallés. "Collision Avoidance of Mobile Robots Using Multi-Agent Systems.". En: *Distributed Computing and Artificial Intelligence. Ed. Springer*, ISBN: 978-3- 319-00550- 8, pp. 4520-437, 2013.

9.3.2 Revistas:

- **A. Soriano**, E. J. Bernabeu, A.Valera, M.Vallés."Simulation, Experimental Implementation and Testing of a Collision Avoidance Method Based on Distributed Consensus Among Mobile Robotic Agents.". En: *International Journal of Imaging & Robotics*, ISSN: 2231-525X, 2015.

- C. Sánchez, **A. Soriano**, M. Vallés, E. Vendrell, A. Valera."$GEMA^2$: Geometrical Matching Analytical Algorithm for Fast Mobile Robots Global Self-Localization.". En: *Robotics & Autonomous Systems*,ISSN: 0929-5593,2014.

- A. Valera, **A. Soriano**, M. Vallés."Plataformas de Bajo Coste para la Realización de Trabajos Prácticos de Mecatrónica y Robótica." En:*Revista Iberoamericana de Automática e Informática industrial*, ISSN:1697-7912, 2014.

- L. Marín, M. Vallés, **A. Soriano**, A. Valera, P. Albertos. "Event-Based Localization in Ackermann Steering Limited Resource Mobile Robots". En: *IEEE-ASME Transactions on Mechatronics*, 2013, Volumen 19 Número 4, páginas 1171-1182, ISSN: 1083-4435, DOI: 10.1109/TMECH. 2013.2277271. Factor de Impacto 3.315, cuartil Q1 (Automation & Control Systems; Electrical & Electronic Engineering; Manufacturing - Mechanical Eng.).

- L. Marín, M. Vallés, **A. Soriano**, A. Valera, P. Albertos. "Multi Sensor Fusion Framework for Indoor-Outdoor Localization of Limited Resource Mobile Robots". En: *Sensors, Special Issue "State-of-the-Art Sensors Technology in Spain 2013"*, 2013, Volumen 13 Número 10, páginas 14133-14160, ISSN 1424-8220, DOI: 10.3390/s131014133. Factor de Impacto 1.953, cuartil Q1 (Instruments & Instrumentation).

9.3.3 Congresos Internacionales:

- **A. Soriano**, E. Bernabeu, Á. Valera, M. Vallés."Multi-Agent Systems for Evasive Maneuvers of Mobile Robots through Agreements." En: *X International Conference on informatics in Control, Automation and Robotics(iCinco)*, 2013.

- **A. Soriano**, L. Marín, M. Vallés, Á. Valera, P. Albertos."Low Cost Platform for Automatic Control Education Based on Open Hardware and Mobile Devices." En: *World Congress of the International Federation of Automatic Control(IFAC)*, 2014.

- A. Valera, M. Valles, L. Marín, **A. Soriano**, A. Cervera, A. Giret. "Application and evaluation of Lego NXT tool for Mobile Robot Control". En: *18th World Congress of the International Federation of Automatic Control (IFAC)*, Milano (Italy), Agosto 28 - Setiembre 2, 2011, ISBN: 978-3-902661-93-7, DOI: 10.3182/20110828-6-IT-1002.01846. Indexado en: SciVerse Scopus database y en IFAC-PapersOnLine Digital Library.

- J. Cazalilla, M. Vallés, M. Díaz-Rodriguez, V.Mata, **A. Soriano**, A. Valera."Implementation of dynamic controllers using real-time middleware for a low-cost parallel robot," En *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

9.3.4 Congresos Nacionales:

- A. Valera, F. Gómez, P. Furió, **A. Soriano**, M. Vallés."Control y coordinación de robots móviles mediante una arquitectura cliente-servidor.". En: *XXXI Jornadas de Automática*, Jaén (España), 2010, ISBN: 978-84- 693-0715- 1.

- A. Cervera, **A. Soriano**, J.Gómez, A.Valera, M.Vallés, A.Giret. "Enfoque holónico basado en agentes para el control de organizaciones de robots móviles.". En: *XXXII Jornadas de Automática*, Sevilla (España), 2011, ISBN: 978-84-694-6454-0.

- **A. Soriano**, L. Marín, J. Gómez-Moreno, A. Valera, M. Vallés, A. Giret. "Organizaciones Holónicas Multiagente Para Resolver Misiones Mediante Robots Móviles.". En: *XXXIII Jornadas de Automática*, Vigo (España), 2012, ISBN: 978-84-8158-583-4.

- J. Gómez-Moreno, **A. Soriano**, M. Vallés, A. Valera, M. Martínez. "Implementación de un algoritmo de localización basado en un método de Montecarlo para un robot móvil omnidireccional.". En: *XXXIII Jornadas de Automática*, Vigo (España), 2012, ISBN: 978-84-8158-583-4.

- **A. Soriano**, L. Marín, R. Juan, J. Cazalilla, A. Valera, M. Vallés, P. Albertos. "Plataforma robótica de bajo coste y recursos limitados basada en Arduino y dispositivos móviles". En: *XXXIV Jornadas de Automática*, Terrassa, Barcelona (España), 2013, ISBN: 978-84-616-5063-7.

- L. Marín, **A. Soriano**, V. Mayans, M. Vallés, A. Valera, P. Albertos. "Localización asistida por GPS para robots móviles en configuración Ackermann de recursos limitados". En: *XXXIV Jornadas de Automática*, Terrassa, Barcelona (España), 2013, ISBN: 978-84-616-5063-7.

- **A. Soriano**, L. Marín, A. Valera, M. Vallés. "Integración de sistemas multiagente en sistemas embebidos con recursos limitados para la realización de tareas de coordinación y cooperación.". En: *XXXV Jornadas de Automática*, Valencia (España), 2014, ISBN: 978-84- 697-0589-6.

- M. Bosch-Jorge, **A. Soriano**, A. Valera, A.Sánchez_salmerón. "Detección y seguimiento de objetos utilizando un escáner láser.". En: *XXXV Jornadas de Automática*, Valencia (España), 2014, ISBN: 978-84- 697-0589-6.

- **A. Soriano**, M. Vallés, A. Valera, P. Albertos "Algoritmo de navegación autónoma basado en una arquitectura distribuida.". En: *XXXV Jornadas de Automática*, Valencia (España), 2014, ISBN: 978-84- 697-0589-6.

- J. Navarro, J. V. Capella, M. Bosch, **A. Soriano**, M. Alvero, A. Valera. "Desarrollo de una plataforma HW/SW para el control de vehículos automá-

ticos.". En: *XXXVI Jornadas de Automática*, Bilbao (España), 2015, ISBN: 978-84-15914-12-9.

Apéndice A

Modelo de referencia basado en la especificación FIPA:

Los componentes básicos que deben formar parte de toda plataforma de agentes según el modelo de referencia de FIPA son los siguientes:

- Agente. Se trata de un proceso computacional que implementa la funcionalidad autónoma y comunicativa de una aplicación. Un agente debe tener, al menos, un propietario y debe poseer un identificador que permita distinguirlo sin ambigüedades del resto de agentes. Dependiendo de la implementación concreta, un agente puede ser un objeto Java, un componente COM, un programa Lisp autocontenido o un script TCL.
- Facilitador de Directorio (Directory Facilitator, DF).Es un componente opcional de la plataforma. Proporciona un servicio de páginas amarillas que permite buscar un agente por sus capacidades, y no sólo por su nombre. Los agentes se registran en el DF indicando los servicios que ofrecen. Cuando otro agente tiene unas necesidades concretas, lanza una búsqueda del servicio deseado obteniendo los agentes que le ofrecen estos servicios.
- Sistema de Gestión de Agentes (Agent Management System, AMS). Es un componente obligatorio de la plataforma. El AMS ejerce un control de supervisión sobre el acceso y uso de la plataforma de agentes. En cada plataforma, sólo puede existir un único AMS. Entre sus responsabilidades, están la creación, destrucción y control del cambio de estado de los agentes, supervisión de los permisos para que nuevos agentes se registren en la plataforma, control de la movilidad de los agentes, gestión de los recursos compartidos y gestión del canal de comunicación (Ana Mas, 2005, p. 52). El AMS ofrece un servicio

de páginas blancas mediante el cual se asocia a cada agente (su identificador único) la dirección de transporte real en la que se encuentra este agente, proporcionando así un método básico para la búsqueda de agentes.

- Sistema de Transporte de Mensajes (Message Transport Service, MTS). Es el método de comunicación por defecto entre agentes de una plataforma y entre agentes de distintas plataformas. Todo agente debe tener acceso a un MTS, ya que éste es el encargado de hacer llegar los mensajes ACL desde su agente origen a su agente destino. El modelo de comunicación entre agentes es asíncrono, lo que implica que el MTS no se queda bloqueado ante el envío o recepción de mensajes y que existen colas de envío y recepción de mensajes (Ana Mas, 2005, p. 53).

- Plataforma de Agentes (Agent Platform, AP). Proporciona la infraestructura física en la cual los agentes pueden ser desplegados. La AP está compuesta por la máquina o máquinas (recursos hardware), el sistema operativo, el software de gestión de agentes, los componentes de gestión de agentes FIPA (DF, AMS y MTS) y los agentes (recursos software), esto es, todo lo necesario para poner en marcha la infraestructura.

- Software Representa a todas las colecciones de instrucciones ejecutables que no son parte de los agentes pero que son accesibles a través de los mismos. Los agentes pueden acceder al software para, por ejemplo, añadir nuevos servicios, incorporar nuevos protocolos de comunicación, procurar nuevos algoritmos y protocolos de seguridad, permitir nuevos protocolos de negociación, etcétera.

Estos componentes se basan en los distintos elementos que se definen en la arquitectura abstracta de FIPA. Los elementos abstractos que son obligatorios en toda implementación de la arquitectura junto con una breve descripción de los mismos se presentan en la tabla siguiente:

Elemento	**Descripción**
Estado-acción	Indicación del estado enviada por un servicio y que muestra si una acción se ha ejecutado con éxito o no.
Agente	Proceso computacional que implementa la funcionalidad autónoma y comunicativa de una aplicación.
Lenguaje comunicación-agente	Lenguaje con sintaxis, semántica y pragmática definidas de forma precisa, el cual es la base de la comunicación entre agentes diseñados y desarrollados independientemente.
Entrada-directorio-agente	Entidad compuesta que contiene el nombre, el localizador del agente y sus atributos.

Continúa en la página siguiente.

Elemento	Descripción
Servicio-directorio-agente	Servicio que provee un repositorio de información compartida en el que las entradas del directorio del agente pueden ser almace nadas y consultadas.
Localizador-agente	Consiste en un conjunto de descripciones sobre el transporte usado para comunicarse con un agente.
Nombre-agente	Un token opaco y no falsificable que identifica de forma unívoca a un agente.
Contenido	La parte del mensaje que representa el componente dependiente del dominio de la comunicación.
Lenguaje-contenido	Un lenguaje para expresar el contenido de una comunicación entre agentes.
Representación- codificación	Modo de representar una sintaxis abstracta en una sintaxis particular (por ejemplo XML, FIPA Strings, objetos Java serializados)
Servicio-codificación	Servicio que codifica un mensaje desde una carga-útil.
Sobre	Parte del mensaje-transporte que contiene información acerca de cómo enviar el mensaje al receptor. Puede incluir información adicional.
Mensaje	Unidad de comunicación entre dos agentes. Se expresa en un lenguajecomunicación-agente y se codifica en una representación-codificación.
Servicio-transporte-mensaje	Servicio que premite enviar y recibir mensaje--transporte entre agentes.
Carga-útil	Un mensaje codificado de un modo apropiado para su inclusión en un mensaje-transporte.
Servicio	Servicio proporcionado por agentes y otros servicios.
Dirección-servicio	Cadena específica para un tipo-servicio que contiene información sobre el direccionamiento del transporte.
Entrada-directorio-servicio	Entidad compuesta que contiene el nombre--servicio, localizador--servicio, y tipo-servicio de un servicio.
Servicio-directorio-servicio	Un servicio de directorio para registrar y descubrir servicios.
Nombre-servicio	Identificador único de un servicio en particular.

Continúa en la página siguiente.

Elemento	**Descripción**
Descipción-localización-servicio	Una tupla clave-valor que contiene un tipo-firma, una firma-servicio y dirección-servicio.
Localizador-servicio	Consiste en un conjunto de descripción-localización-servicio usados para acceder a un servicio.
Raíz-servicio	Conjunto de entrada-directorio-servicio.
Firma-servicio	Identificador que describe la firma de comprobación para un servicio.
Tipo-servicio	Tupla clave-valor describiendo el tipo de un servicio.
Transporte	Servicio de entrega de datos sustentado por un servicio-transporte-mensaje.
Descripción-transporte	Estructura auto-descriptiva que contiene tipo--transporte, dirección-específica-transporte y cero o más propiedad-específica-transporte.
Mensaje-transporte	Objeto transmitido de agente a agente. Contiene la descripción-transporte para el remitente y receptor o receptores, junto con una carga-útil que contiene el mensaje.
Dirección-específica-transporte	Dirección de transporte específica para un tipo--transporte dado.
Tipo-transporte	Describe el tipo de transporte asociado con una dirección-específica-transporte.

Tabla A.1: Descripción de los elementos obligatorios de la Arquitectura Abstracta de FIPA

Apéndice B

Modelos Cinemático y Dinámico de los Robots

Un componente fundamental a la hora de controlar cualquier sistema correctamente es la obtención de un modelo fiable de dicho sistema. Para el caso concreto de la localización de un robot móvil, es imprescindible la obtención de un modelo cinemático y dinámico del robot que logre caracterizar correctamente su comportamiento o su movimiento. Para este trabajo, se ha utilizado un enfoque basado en la equivalencia dinámica entre los sistemas mecánicos y los sistemas de partículas con el fin de simplificar el proceso y facilitar su implementación en plataformas de recursos limitados. El presente anexo describe los modelos utilizados para la configuración diferencial y la configuración Ackerman.

B.1 Configuración diferencial

Un robot en configuración diferencial consiste en un cuerpo rígido de masa M_G y un momento de inercia I_G con dos ruedas no deformables y no orientables (fijas) separadas una distancia b y que son controladas por dos motores que aplican dos fuerzas lineales F_R y F_L que producen las aceleraciones a_R y a_L tal como se muestra en la figura B.1. El robot tiene la restricción de moverse únicamente en un plano horizontal y su centro de masa, denominado P_0, se corresponde con su centro de gravedad y con su eje de rotación. Para simplificar el proceso de obtención del modelo, se asumen ciertas condiciones ideales como que las ruedas son convencionales, no deformables y que satisfacen la condición de rodamiento puro sin deslizamiento. Además se considera que cumple la condición de movimiento para bajas velocidades.

B.1.1 Modelo cinemático

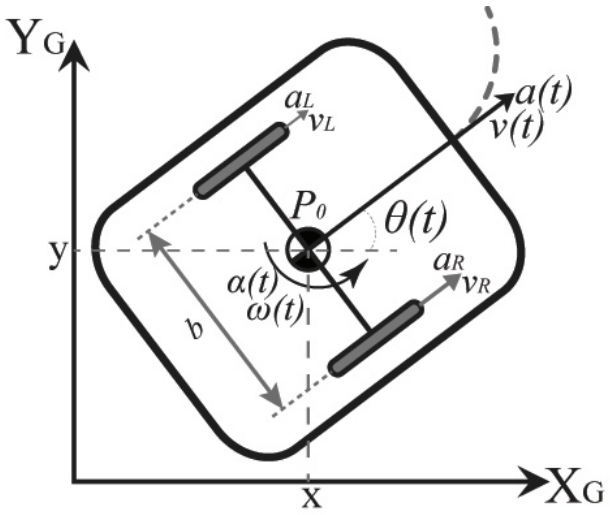


Figura B.1: Modelo cinemático de la configuración diferencial.

El modelo cinemático representa la evolución de las velocidades del robot en un marco de referencia inercial fijo. La *postura* o *pose* del robot se define por su posición (x, y) (coordenadas del punto P_0) y por su ángulo de orientación θ (también denominado ángulo de avance, curso del robot o guiñada) en el marco de referencia *Global* (X_G,Y_G) mostrado en la figura B.1.

Dada una conocida velocidad lineal y angular (v y ω) en el marco de referencia *Local* (X_L,Y_L), la velocidad global del robot se define como:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \Rightarrow \begin{array}{c} \dot{x}(t) = v(t)\cos(\theta(t)) \\ \dot{y}(t) = v(t)\sin(\theta(t)) \\ \dot{\theta}(t) = \omega(t) \end{array} \tag{B.1}$$

En la literatura se encuentran diversas maneras y métodos de resolver esta ecuación diferencial, como la integración directa o las aproximaciones numéricas tal y como se exponen en [186], [50] y [9]). Dado que en este caso se pretende simplificar el tiempo de ejecución del algoritmo, es conveniente obtener el resultado exacto de la integral y evaluar el uso de las aproximaciones que se consideren convenientes.

Para obtener la solución exacta a la ecuación (B.1), se pretende resolver de forma discreta para un periodo de muestreo $T_s = t_{k+1} - t_k$ de manera que se obtenga una ecuación recursiva que pueda implementarse dentro del robot. De esta forma se

define el intervalo de tiempo entre dos instantes consecutivos de muestreo mostrado en la siguiente ecuación:

$$T_K \;=\; \{t \in \mathbb{R} \mid kT_s \;\leqslant t \;<\; (k\;+1)\,T_s\;\},\; k \;=\; 0,1,2,\ldots \tag{B.2}$$

Se define $t_k = \; kT_s$ y $t_{k+1} = (k\;+1)\,T_s$ con el objetivo de simplificar la deducción de las ecuaciones. Además se considera que las velocidades (v, ω) son constantes en el intervalo de integración (no cambian mientras se realiza el muestreo en T_s, lo que es equivalente a decir que se tiene un retenedor de orden cero (ZOH) en estas entradas, adquiriendo el valor en $t_k \Rightarrow v\,(t_k)\,, \omega\,(t_k)$ como constantes. De esta forma se plantea la integral en (B.3).

$$\begin{aligned} x\,(t) &= x\,(t_k) + v\,(t_k) \int_{t_k}^{t} \cos\left(\theta\,(\tau)\right) d\tau \\ y\,(t) &= y\,(t_k) + v\,(t_k) \int_{t_k}^{t} \sin\left(\theta\,(\tau)\right) d\tau \\ \theta\,(t) &= \theta\,(t_k) + \int_{t_k}^{t} \omega\,(t_k)\, d\tau \end{aligned} \tag{B.3}$$

El ángulo de avance del robot θ se puede obtener como:

$$\theta\,(t) = \theta\,(t_k) + \int_{t_k}^{t} \omega\,(t_k)\, d\tau = \theta\,(t_k) + \omega\,(t_k) \int_{t_k}^{t} d\tau = \theta\,(t_k) + (t - t_k)\,\omega\,(t_k) \tag{B.4}$$

En la ecuación (B.3), se sustituye el resultado anterior para las ecuaciones de x e y. Cabe destacar que las funciones evaluadas en t_k son constantes $(x\,(t_k)\,, y\,(t_k)\,, \theta\,(t_k))$ ya que son las condiciones iniciales (constantes de integración). Una vez realizada la sustitución, se procede a integrar (B.3) dando el resultado de la ecuación (B.5) para x y de la ecuación (B.6) para y.

$$
\begin{aligned}
x(t) &= x(t_k) + v(t_k) \int_{t_k}^{t} \cos\left(\theta(t_k) + (\tau - t_k)\,\omega(t_k)\right) d\tau \\
&= x(t_k) + \frac{v(t_k)}{\omega(t_k)} \left[\sin\left(\theta(t_k) + (\tau - t_k)\,\omega(t_k)\right)\right]_{t_k}^{t} \\
\Rightarrow x(t) &= x(t_k) + \frac{v(t_k)}{\omega(t_k)} \left[\sin\left(\theta(t_k) + (t - t_k)\,\omega(t_k)\right) - \sin\left(\theta(t_k) + (t_k - t_k)\,\omega(t_k)\right)\right] \\
\therefore x(t) &= x(t_k) + \frac{v(t_k)}{\omega(t_k)} \left[-\sin\left(\theta(t_k)\right) + \sin\left(\theta(t_k) + (t - t_k)\,\omega(t_k)\right)\right]
\end{aligned}
\tag{B.5}
$$

$$
\begin{aligned}
y(t) &= y(t_k) + v(t_k) \int_{t_k}^{t} \sin\left(\theta(t_k) + (\tau - t_k)\,\omega(t_k)\right) d\tau \\
&= y(t_k) - \frac{v(t_k)}{\omega(t_k)} \left[\cos\left(\theta(t_k) + (\tau - t_k)\,\omega(t_k)\right)\right]_{t_k}^{t} \\
\Rightarrow y(t) &= y(t_k) - \frac{v(t_k)}{\omega(t_k)} \left[\cos\left(\theta(t_k) + (t - t_k)\,\omega(t_k)\right) - \cos\left(\theta(t_k) + (t_k - t_k)\,\omega(t_k)\right)\right] \\
\therefore y(t) &= y(t_k) + \frac{v(t_k)}{\omega(t_k)} \left[\cos\left(\theta(t_k)\right) - \cos\left(\theta(t_k) + (t - t_k)\,\omega(t_k)\right)\right]
\end{aligned}
\tag{B.6}
$$

Las expresiones (B.5) y (B.6) se pueden simplificar aplicando identidades trigonométricas[1], tal y como se muestra en (B.7):

$$
\left.\begin{array}{c} a = \theta(t_k) \\ b = (t - t_k)\,\omega(t_k) \end{array}\right\} \Rightarrow \begin{array}{l} -\sin\left(\theta(t_k)\right) + \sin\left(\theta(t_k) + (t - t_k)\,\omega(t_k)\right) = \sin(a+b) - \sin(a) \\ \cos\left(\theta(t_k)\right) - \cos\left(\theta(t_k) + (t - t_k)\,\omega(t_k)\right) = \cos(a) - \cos(a+b) \end{array}
$$

$$
\begin{aligned}
&\sin(a+b) - \sin(a) = 2\sin(0.5\,(a+b-a))\cos(0.5\,(a+b+a)) = 2\sin(0.5b)\cos(0.5\,(2a+b)) \\
&\cos(a) - \cos(a+b) = -2\sin(0.5\,(a-a-b))\sin(0.5\,(a+b+a)) = -2\sin(-0.5b)\sin(0.5\,(2a+b))
\end{aligned}
\tag{B.7}
$$

[1] $\sin u - \sin v = 2\cos(0.5u + 0.5v)\sin(0.5u - 0.5v)$,
$\cos u - \cos v = -2\sin(0.5u + 0.5v)\sin(0.5u - 0.5v)$, $\sin(-u) = -\sin(u)$

Aplicando esta simplificación a las ecuaciones (B.5) y (B.6) se obtiene:

$$\begin{aligned}
&\Rightarrow x(t) = x(t_k) + 2\frac{v(t_k)}{\omega(t_k)}\sin(0.5(t-t_k)\omega(t_k))\cos(\theta(t_k) + 0.5(t-t_k)\omega(t_k)) \\
&\Rightarrow y(t) = y(t_k) + 2\frac{v(t_k)}{\omega(t_k)}\sin(0.5(t-t_k)\omega(t_k))\sin(\theta(t_k) + 0.5(t-t_k)\omega(t_k)) \\
&\theta(t) = \theta(t_k) + (t-t_k)\omega(t_k)
\end{aligned} \tag{B.8}$$

Este es el resultado completo de la integración para cualquier $t \in T_K$ y evaluando la ecuación (B.8) en $t = t_{k+1}$ se obtiene el modelo discreto en el periodo de muestreo $T_s = t_{k+1} - t_k$ tal y como se muestra a continuación:

$$\begin{aligned}
x(t_{k+1}) &= x(t_k) + 2\frac{v(t_k)}{\omega(t_k)}\sin(0.5(t_{k+1}-t_k)\omega(t_k))\cos(\theta(t_k) + 0.5(t_{k+1}-t_k)\omega(t_k)) \\
y(t_{k+1}) &= y(t_k) + 2\frac{v(t_k)}{\omega(t_k)}\sin(0.5(t_{k+1}-t_k)\omega(t_k))\sin(\theta(t_k) + 0.5(t_{k+1}-t_k)\omega(t_k)) \\
\theta(t_{k+1}) &= \theta(t_k) + (t_{k+1}-t_k)\omega(t_k)
\end{aligned} \tag{B.9}$$

Por último, si se sustituye el periodo de muestreo $T_s = t_{k+1} - t_k$ en (B.9) y se simplifica la notación de modo que $x(t_k) = x_k$, se obtiene la expresión del modelo cinemático del robot de la ecuación (B.10). Esta expresión puede considerarse como un simple cambio de coordenadas ya que relaciona el movimiento local del robot (v, ω) con su movimiento global (x, y, θ). Para facilitar su comprensión se muestra a continuación la notación que se ha utilizado en los algoritmos de fusión desarrollados, tomando en cuenta el instante actual k y el anterior $k-1$.

$$\begin{aligned}
\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} &= \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} \frac{2v_k}{\omega_k}\sin(0,5T_s\omega_k)\cos(\theta_k + 0,5T_s\omega_k) \\ \frac{2v_k}{\omega_k}\sin(0,5T_s\omega_k)\sin(\theta_k + 0,5T_s\omega_k) \\ T_s\omega_k \end{bmatrix} = \\
\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} &= \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{2v_{k-1}}{\omega_{k-1}}\sin(0,5T_s\omega_{k-1})\cos(\theta_{k-1} + 0,5T_s\omega_{k-1}) \\ \frac{2v_{k-1}}{\omega_{k-1}}\sin(0,5T_s\omega_{k-1})\sin(\theta_{k-1} + 0,5T_s\omega_{k-1}) \\ T_s\omega_{k-1} \end{bmatrix}
\end{aligned} \tag{B.10}$$

Teniendo en cuenta que el periodo de muestreo con el que se va a trabajar va a ser alto para realizar una implementación estable del control del robot, (T_s muy pequeño) se puede realizar bajo esta premisa una aproximación en la integral de la ecuación diferencial (B.10) considerando que $\sin(\omega_k \cdot 0,5T_s) \approx 0,5T_s\omega_k$ tal y como se muestra en (B.11).

$$L_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + T_s \begin{bmatrix} v_{k-1}\cos\left(\theta_{k-1} + \omega_{k-1} \cdot 0,5T_s\right) \\ v_{k-1}\sin\left(\theta_{k-1} + \omega_{k-1} \cdot 0,5T_s\right) \\ \omega_{k-1} \end{bmatrix} \tag{B.11}$$

Considerando además, que el robot gira a una velocidad tal que para un tiempo T_s el incremento de θ_k es muy pequeño, entonces la ecuación (B.11) se puede simplificar aún más ya que en estas condiciones $\theta_k + \omega_k \cdot 0,5T_s \approx \theta_k$, tal y como se muestra en (B.12).

$$L_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + T_s \begin{bmatrix} v_{k-1}\cos\left(\theta_{k-1}\right) \\ v_{k-1}\sin\left(\theta_{k-1}\right) \\ \omega_{k-1} \end{bmatrix} \tag{B.12}$$

Las dos ecuaciones anteriores, (B.11) y (B.12) son ampliamente usadas en la literatura ([121] o [163]) sin discriminar las condiciones necesarias para que éstas estimen de forma adecuada la *pose* del robot, ni se establecen los requerimientos del robot para poder resolver esta ecuación sin superar el tiempo de muestreo T_s. Se puede observar que la ecuación (B.12) es más simple y ligera que la ecuación (B.11), por lo que requiere un tiempo de cómputo menor y facilita que no se sobrepase el T_s en robots con una capacidad de cómputo muy limitada. Por este motivo, en la práctica se ha implementado la ecuación (B.12) en aquellos robots con una capacidad de cómputo tan limitada que no eran capaces de cumplir con el periodo de muestreo T_s cuando ejecutaban la ecuación (B.11).

Para finalizar con el modelado cinemático de la configuración diferencial, cabe destacar la relación existente entre v y ω y las velocidades lineales de las ruedas (v_L,v_R obtenidas a partir del cambio en el desplazamiento de los encoders) tal como se muestra en (B.13). Derivando esta ecuación, es posible obtener una relación cinemática entre las aceleraciones lineales y angulares a y α del robot y las aceleraciones de las ruedas izquierda a_L y derecha a_R. Como se muestra más adelante, en la implementación del modelo se han utilizado dos acelerómetros colocados sobre cada rueda para obtener otra medición de la velocidad del robot además de la calculada a partir de los encoders. Estas aceleraciones medidas por los sensores, se han integrado con la intención de obtener las velocidades y usarlas como una entrada más en el filtro de fusión para obtener la posición global del robot.

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ -1/b & 1/b \end{bmatrix} \begin{bmatrix} v_L \\ v_R \end{bmatrix} \Rightarrow \begin{bmatrix} a \\ \alpha \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ -1/b & 1/b \end{bmatrix} \begin{bmatrix} \dot{v}_L \\ \dot{v}_R \end{bmatrix} \tag{B.13}$$

Otro modo de obtener dichas aceleraciones es hacer uso de la dinámica del robot, tal y como se expone en la siguiente sección.

B.1.2 Modelo dinámico

En la literatura existen una gran variedad de desarrollos para la obtención del modelo dinámico de un robot en configuración diferencial. Un modelo dinámico representa la evolución de la aceleración lineal y angular del robot (a,α) en términos de fuerza ejercida por cada motor $F_{L,R}$ y su momento angular o torque de rotación τ. El procedimiento más utilizado para su obtención consiste en aplicar la segunda Ley de Newton para obtener los valores de a y α en el centro de masas del robot de manera local (en cada motor o rueda) y junto al modelo cinemático obtener la aceleración global del robot. Este procedimiento se detalla en trabajos como [38, 51, 73, 188, 51], y [2]). En otros trabajos como por ejemplo [156, 192, 130] y [200], se aplica la formulación de Lagrange para obtener directamente el modelo en coordenadas globales del robot sin necesidad del modelo cinemático. Por lo general, el resultado que ofrecen estas dos estrategias suelen ser modelos teóricos complejos que difícilmente son aplicables a la práctica debido a la complejidad de identificar u obtener la mayoría de los parámetros que se requieren de forma experimental. Otra gran desventaja de estas estrategias consiste en que se obtienen como resultado modelos no lineales de dimensión alta, por lo que las versiones de los filtros de fusión sensorial aplicables se reducen al EKF o al UKF. Estas versiones de filtros requieren el cálculo de inversión de matrices de gran dimensión y el cálculo de raíces de orden alto, por lo que su implementación en robots de recursos limitados resulta inviable en la práctica.

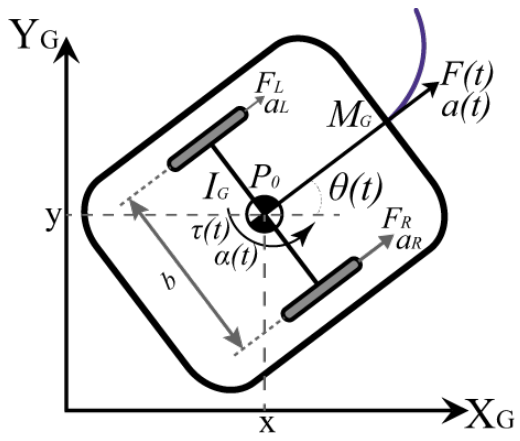


Figura B.2: Modelo dinámico de un robot de configuración diferencial

El motivo por el cual estas estrategias devuelven modelos tan complejos es que para obtener el movimiento del robot en el marco de referencia *global*, es decir, en los ejes X_G, Y_G mostrados en la figura B.2), se parte de la aceleración *local* del robot (lineal a y angular α) y se realiza el cálculo del seno y coseno del ángulo de avance del robot θ, de forma similar a la derivada del modelo cinemático de la ecuación

(B.1). Con esto se obtiene la aceleración en los ejes globales la cual es integrada dos veces para obtener el modelo de la *pose* del robot (x, y, θ). Esta transformación de coordenadas es la principal fuente de no linealidad del modelo (excluyendo los más complejos que estiman el deslizamiento en las ruedas del robot). Ante este problema se plantea como solución el utilizar un modelo dinámico *local* para realizar la fusión sensorial (por ejemplo, utilizando un filtro de Kalman) y posteriormente aplicar la transformación de coordenadas (modelo cinemático) para obtener la dinámica *global*. Si bien es cierto que esta solución es en apariencia trivial, no lo es, ya que con esto se obtiene una ventaja fundamental, se logra simplificar el cálculo del método de fusión ya que se utiliza un modelo de menores dimensiones (y en algunos casos lineal) para estimar la dinámica local, incorporando los distintos sensores de la plataforma para así estimar la *pose* del robot mediante la ecuación (B.11).

Además de las dificultades comentadas en el párrafo anterior, existe otro problema de índole práctico: la obtención del modelo dinámico local para calcular la aceleración del robot (a y α) cuando se requiere conocer la fuerza que ejercen los motores, no es sencilla de resolver. En la mayoría de plataformas experimentales utilizadas en esta tesis no es posible medir el par del motor, ni su consumo de corriente y ni siquiera en los robots comerciales utilizados se conocen los momentos de inercia o los coeficientes de amortiguamiento, por lo que obtener un modelo adecuado y fiable se convierte en una tarea ardua de resolver.

Para solventar esta cuestión y poder obtener los modelos de las distintas plataformas experimentales, se ha obtenido un modelo dinámico de implementación sencilla el cual no depende de las fuerzas de las ruedas sino de la aceleración de las mismas. Estas aceleraciones se han obtenido a partir de la medición directa de un acelerómetro 3D instalado justo encima de cada rueda del robot. En los casos en los que no se disponía de acelerómetro, se ha utilizado un modelo que relaciona la acción de control aplicada a cada rueda con la aceleración resultante. El procedimiento aplicado en general, consiste en obtener el modelo dinámico utilizando las fuerzas como entradas mediante la segunda Ley de Newton. Posteriormente se determina el modelo dinámico del movimiento local del robot que depende de las aceleraciones de las ruedas (a_L,a_R), mediante un sistema de partículas dinámicamente equivalente en versiones de dos y tres partículas. Finalmente, se obtiene el modelo dinámico de las aceleraciones de las ruedas del robot según la información devuelta por los encoders y la acción de control del robot.

A continuación se detalla el fundamento de la estrategía de la obtención del modelo dinámico basado en fuerzas según la segunda Ley de Newton, así como la obtención del sistema de partículas dinámicamente equivalente para sus versiones de dos y tres partículas.

B.1.2.1 Cálculo del Modelo Dinámico basado en Fuerzas

Dado el cuerpo rígido de un robot de masa M_G y su momento de inercia I_G, si se aplican dos fuerzas lineales (F_L,F_R) una en cada rueda (fuerza resultante del motor menos la fricción), estando separadas las mismas a una distancia b entre sí tal y como se muestra en la figura B.3, estas fuerzas con signo variable son perpendiculares al eje de las ruedas y generan una fuerza resultante de traslación (F) y un par de rotación (τ) sobre el centro de masa considerado como P_0. Si el centro de masa es distinto al del punto P_0 pero desplazado en la dirección de F, las fuerzas pueden trasladarse sobre sus ejes hasta coincidir con este punto, por lo que el análisis es válido para ambos casos.

Al aplicar la segunda ley de Newton al cuerpo rígido del robot se obtiene la ecuación (B.14) donde quedan reflejadas las fuerzas en cada rueda. Sin embargo, para la obtención del modelo es necesario el cálculo de sus aceleraciones por lo que se ha desarrollado un sistema de partículas dinámicamente equivalente en dos versiones, una de dos partículas y otra de tres.

$$\begin{array}{l} \sum F = F_R + F_L = M_G a \\ \sum \tau = \tau_R - \tau_L = 0.5b\,(F_R - F_L) = I_G \alpha \end{array} \tag{B.14}$$

B.1.2.2 Sistema de partículas dinámicamente equivalente de dos partículas

Para obtener las aceleraciones de las ruedas a partir de las fuerzas de las ecuaciones (B.14), es necesario conocer la *masa* que transporta cada rueda. En los casos donde cada rueda se mueve a la misma velocidad, es plausible suponer que cada una desplaza la mitad de la masa del robot (asumiendo una densidad constante), pero en los casos donde cada una gira a una velocidad o incluso una de las dos está frenada y la otra no, dicha suposición no es correcta. Es por esto por lo que se busca el desarrollo de un sistema de dos partículas dinámicamente equivalente.

Bajo el mismo objetivo, en [14] se aplican los sistemas de partículas a casos prácticos de brazos manipuladores 3D. En este trabajo se ha extendido la misma estrategia para robots móviles de configuración diferencial, considerando el cuerpo rígido del robot de masa M_G y con un momento de inercia I_G, como un sistema equivalente de dos partículas de masa M_L y M_R unidas por un conector o una barra sin masa de longitud constante $R_L + R_R$ tal y como muestra la figura B.3.

En [14] también se exponen las condiciones necesarias para poder aplicar este tipo de metodología:

- Conservación de la masa: *La suma de las masas de las partículas debe ser igual a la masa del cuerpo rígido*: $M_G = M_L + M_R$

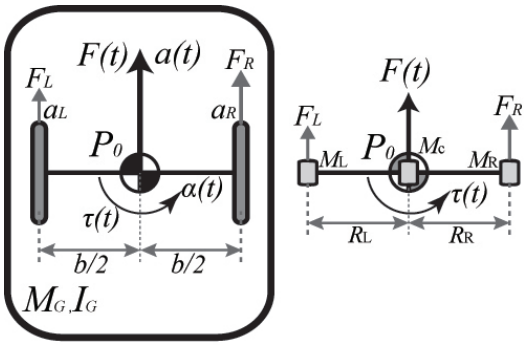


Figura B.3: Modelo dinámico basado en fuerzas de un robot de configuración diferencial.

- Conservación del centro de masa: *Las masas deben estar alineadas con el centro de masas y una a cada lado de este centro*: $M_L R_L = M_R R_R$
- Conservación del momento de inercia: *La suma de los momentos de inercia de las partículas debe ser igual al momento de inercia del cuerpo rígido*: $M_L R_L^2 + M_R R_R^2 = I_G$

Teniendo en cuenta estas condiciones y el modelo dinámico basado en fuerzas, se despejan las masas y radios del sistema equivalente tal y como se muestra en la ecuación (B.15).

$$\begin{aligned}
&M_L R_L = M_R R_R \Rightarrow M_R = \frac{R_L}{R_R} M_L \\
&M_L R_L^2 + M_R R_R^2 = I_G \Rightarrow M_L R_L^2 + \frac{R_L}{R_R} R_R^2 M_L = I_G \\
&\Rightarrow M_L R_L^2 + M_L R_L R_R = I_G \Rightarrow M_L R_L \left(R_L + R_R\right) = I_G \\
&\Rightarrow M_L = \frac{I_G}{R_L \left(R_L + R_R\right)} \Rightarrow M_R = \frac{R_L}{R_R} M_L = \frac{I_G}{R_R \left(R_L + R_R\right)} \\
&\Rightarrow M_G = M_L + M_R = \frac{I_G}{\left(R_L + R_R\right)} \left(\frac{1}{R_L} + \frac{1}{R_R}\right) = \frac{I_G}{R_L R_R} \\
&\therefore M_L = \frac{I_G}{R_L \left(R_L + R_R\right)}, M_R = \frac{I_G}{R_R \left(R_L + R_R\right)}, R_L R_R = \frac{I_G}{M_G}
\end{aligned} \tag{B.15}$$

Como se puede observar, las distancias entre las masas dependen entre sí. Por simplicidad, si se suponen equidistantes al centro del robot ($R_L = R_R = R_N$) es posible reducir la ecuación (B.15) a (B.16).

$$M_L = M_R = \frac{I_G}{2R_N^2}, R_N^2 = \frac{I_G}{M_G} \tag{B.16}$$

Las masas M_L y M_R son proporcionales a la masa total M_G, por lo que es posible expresar dicha relación mediante una constante denominada γ. Adicionalmente, es posible definir un factor λ como la razón entre la masa y el momento de inercia del cuerpo rígido multiplicado por la distancia R_N entre el centro de masa y la partícula asignada a cada rueda, tal y como se muestra en la ecuación (B.17).

$$\begin{aligned} M_L = M_R &= \gamma M_G \\ R_N^2 &= \frac{I_G}{M_G} \\ \lambda &= \frac{M_G R_N}{I_G} \end{aligned} \tag{B.17}$$

Estos parámetros de la ecuación (B.17) son dependientes de las formas de los robots según el valor correspondiente de I_G. A continuación se exponen los distintos parámetros según el modelo aplicado.

Para un anillo delgado $I_G = M_G R^2$, luego:

$$\begin{aligned} R_N^2 &= \frac{I_G}{M_G} = \frac{M_G R^2}{M_G} \Rightarrow R_N = R \\ M_L = M_R &= \frac{I_G}{2R_N^2} = \frac{M_G R^2}{2R^2} = 0,5 M_G \\ \lambda &= \frac{M_G R_N}{I_G} = \frac{M_G R}{M_G R^2} = \frac{1}{R} \end{aligned} \tag{B.18}$$

Para un cilindro $I_G = 0.5 M_G R^2$, por tanto:

$$\begin{aligned} R_N^2 &= \frac{I_G}{M_G} = \frac{0.5 M_G R^2}{M_G} \Rightarrow R_N^2 = 0.5R^2 \Rightarrow R_N = \sqrt{0.5}R \\ M_L = M_R &= \frac{I_G}{2R_N^2} = \frac{0.5 M_G R^2}{2(0.5R^2)} = 0,5 M_G \\ \lambda &= \frac{M_G R_N}{I_G} = \frac{M_G \sqrt{0.5} R}{0.5 M_G R^2} = \frac{2\sqrt{0.5}}{R} = \frac{1}{R\sqrt{0.5}} \end{aligned} \tag{B.19}$$

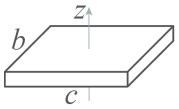
Para un prisma $I_G = \frac{1}{12}\left(b^2 + c^2\right) M_G = \beta M_G, \beta = \frac{1}{12}\left(b^2 + c^2\right)$:

$$\begin{aligned} R_N^2 &= \frac{I_G}{M_G} = \frac{\beta M_G}{M_G} \Rightarrow R_N^2 = \beta \Rightarrow R_N = \sqrt{\beta} \\ M_L &= M_R = \frac{I_G}{2R_N^2} = \frac{\beta M_G}{2\beta} = 0,5 M_G \\ \lambda &= \frac{M_G R_N}{I_G} = \frac{M_G \sqrt{\beta}}{\beta M_G} = \frac{\sqrt{\beta}}{\beta} \end{aligned} \tag{B.20}$$

Conociendo el valor correspondiente de I_G, ya sea experimentalmente o de manera teórica ofrecida para alguna plataforma comercial, es posible obtener los valores numéricos de los parámetros en (B.17) al sustituir los valores de M_G e I_G. Algunos robots comerciales incluyen un modelo CAD (en Solidworks® o similar) los cuales permiten determinar estos parámetros con precisión ante distintas modificaciones físicas de la plataforma.

En resumen, la tabla B.1 muestra los parámetros (γ, λ, R_N) del sistema de dos partículas, calculados para las formas genéricas de un robot diferencial.

Tabla B.1: Parámetros del sistema de partículas dinámicamente equivalente, caso dos partículas para el robot diferencial

Forma del Robot	Momento de Inercia	γ	R_N	λ
Anillo Delgado equivalente a (B.13) 	$I_G = M_G R^2$	0.5	R	$\frac{1}{R}$
Sólido Cilíndrico 	$I_G = \frac{M_G R^2}{2}$	0.5	$R\sqrt{0.5}$	$\frac{1}{R\sqrt{0.5}}$
Sólido Rectangular 	$I_G = \beta M_G$ $\beta = \frac{1}{12}\left(b^2 + c^2\right)$	0.5	$\sqrt{\beta}$	$\frac{\sqrt{\beta}}{\beta}$

Una vez obtenidos estos parámetros ya es posible definir el modelo dinámico *local* del robot aplicando en primer lugar la segunda Ley de Newton al sistema de partículas de la figura B.3. De este modo se obtiene el siguiente modelo:

$$\begin{aligned} \sum F = F_R + F_L \Rightarrow M_R a_R + M_L a_L = M_G a \\ \sum \tau = \tau_R - \tau_L \Rightarrow R_N \left(M_R a_R - M_L a_L \right) = I_G \alpha \end{aligned} \tag{B.21}$$

De este modo se puede afirmar que el sistema dinámico del cuerpo rígido (ecuación (B.14)) y el de partículas (ecuación (B.21)) son dinámicamente equivalentes cuando el sistema de partículas se define utilizando (B.17) y los parámetros de la tabla B.1. Sin embargo, además de estas condiciones, las fuerzas y pares aplicados en (B.14) y (B.21) deben ser los mismos. En el caso del modelo de fuerzas sí se cumple esta condición pero para que los pares sean equivalentes, las distancias deben cumplir con la condición $b = 2R_N$. Bajo esta premisa y utilizando las masas equivalentes en (B.21), se despejan las aceleraciones locales a y α tal y como se observa en (B.22).

$$\begin{aligned} a = \gamma \frac{M_G}{M_G} \left(a_R + a_L \right) \Rightarrow a = \gamma \left(a_R + a_L \right) \\ \alpha = \gamma \frac{M_G R_N}{I_G} (a_R - a_L) \Rightarrow \alpha = \lambda \gamma \left(a_R - a_L \right) \end{aligned} \tag{B.22}$$

Integrando las aceleraciones de forma discreta (el modelo cinemático $a = \dot{v}$,$\alpha = \dot{\omega}$) y sustituyendo en las entradas (aceleraciones) de (B.22), se obtiene el modelo dinámico *local* tal y como se muestra en (B.23), observándose que esta ecuación es lineal e implementable en sistemas de recursos limitados.

$$\begin{bmatrix} v_k \\ \omega_k \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_{k-1} \\ \omega_{k-1} \end{bmatrix} + \begin{bmatrix} T_s & 0 \\ 0 & T_s \end{bmatrix} \begin{bmatrix} a_{k-1} \\ \alpha_{k-1} \end{bmatrix} \tag{B.23}$$

Si se sustituye (B.23) en (B.11) se obtiene el modelo dinámico global del robot tal y como se muestra en (B.24), siendo posible observar que este modelo es no lineal.

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \\ v_k \\ \omega_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \\ v_{k-1} \\ \omega_{k-1} \end{bmatrix} + \begin{bmatrix} v_{k-1} T_s \cos \left(\theta_{k-1} + 0.5 T_s \omega_{k-1} \right) \\ v_{k-1} T_s \sin \left(\theta_{k-1} + 0.5 T_s \omega_{k-1} \right) \\ T_s \omega_{k-1} \\ T_s a_{k-1} \\ T_s \alpha_{k-1} \end{bmatrix} \tag{B.24}$$

Los modelos (B.22), (B.23) y (B.24) se utilizan en este trabajo junto con el método de fusión sensorial para estimar la pose global del robot en las plataformas experimentales donde se ha podido ajustar su diseño para que la distancia entre las ruedas

cumpla la condición de $b = 2R_N$. Esto ocurre en robots de estructura configurable como los LEGO NXT. Sin embargo, otras plataformas comerciales utilizadas no permiten esta reestructuración y no es posible cumplir dicha condición. Para estos casos se ha obtenido un modelo similar utilizando una equivalencia dinámica mediante un sistema de tres partículas, tal y como se describe a continuación.

B.1.2.3 Sistema de tres partículas dinámicamente equivalente

Para evitar la restricción descrita en la sección anterior, es necesario introducir una tercera partícula situada justo en el centro de masas del cuerpo rígido tal y como representa la imagen B.4.

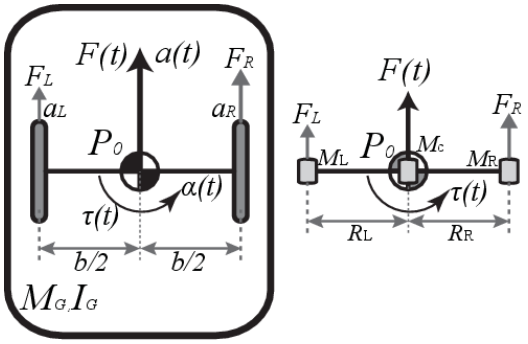


Figura B.4: Modelo dinámico de un robot en configuración diferencial como un cuerpo rígido representado como un sistema equivalente de tres partículas

En este sistema, las tres partículas con masa M_L, M_c y M_R están unidas por dos conectores o barras sin masa de longitud constante R_L y R_R. De nuevo, es necesario extender las condiciones expuestas en [14] pero ahora para el caso de tres partículas:

- Conservación de la masa: $M_G = M_L + M_c + M_R$
- Conservación del centro de masa: $M_L R_L = M_R R_R$
- Conservación del momento de inercia: $M_L R_L^2 + M_R R_R^2 = I_G$

Partiendo de estas condiciones, se despejan las masas del sistema equivalente tal y como se muestra en la ecuación (B.25).

$$
\begin{aligned}
M_G = M_L + M_c + M_R \Rightarrow M_c = M_G - M_L - M_R \\
M_L R_L = M_R R_R \Rightarrow M_R = \frac{R_L}{R_R} M_L \\
M_L R_L^2 + M_R R_R^2 = I_G \\
\Rightarrow M_L R_L^2 + \frac{R_L}{R_R} R_R^2 M_L = I_G \Rightarrow M_L R_L \left(R_L + R_R\right) = I_G \\
\Rightarrow M_L = \frac{I_G}{R_L \left(R_L + R_R\right)} \Rightarrow M_R = \frac{R_L}{R_R} M_L = \frac{I_G}{R_R \left(R_L + R_R\right)} \\
\Rightarrow M_c = M_G - M_L - M_R \\
\Rightarrow M_c = M_G - \left(M_L + M_R\right) = M_G - \left(\frac{I_G}{R_L R_R}\right) \\
\therefore M_L = \frac{I_G}{R_L \left(R_L + R_R\right)}, M_R = \frac{I_G}{R_R \left(R_L + R_R\right)}, M_c = M_G - \left(\frac{I_G}{R_L R_R}\right)
\end{aligned}
\tag{B.25}
$$

A diferencia del sistema equivalente de dos partículas del apartado anterior, en este caso las distancias R_R y R_L sí que son independientes entre sí tal y como demuestra la ecuación (B.25). Por simplicidad en la obtención del modelo,se puede suponer que dichas distancias son iguales ($R_R = R_L = R_N$), por lo que la ecuación anterior se reduciría a la (B.26) expuesta a continuación.

$$
\begin{aligned}
M_L = M_R = \tfrac{I_G}{2R_N^2} = \gamma M_G, \qquad \lambda_a = \tfrac{\gamma}{1-\delta} \\
M_c = M_G - \left(\tfrac{I_G}{R_N^2}\right) = \delta M_G, \quad \lambda_\alpha = \tfrac{M_G R_N}{I_G}
\end{aligned}
\tag{B.26}
$$

Las masas M_L y M_R son proporcionales a la masa total M_G, lo cual se representa mediante la constante γ ($0 \leq \gamma < 1$), y de la misma forma M_c es proporcional a M_G, expresado por la constante δ ($0 \leq \delta < 1$). Adicionalmente se han definido dos constantes, λ_a utilizando γ y δ, y λ_α mediante M_G, R_N e I_G.

Del mismo modo que se desarrolló para el sistema de dos partículas los casos del anillo delgado, el cilíndro y el prisma, se procede previamente con el análisis de las masas utilizando la ecuación (B.26). Por conveniencia y no por necesidad como ocurría en el caso anterior, se considera que las partículas de los extremos están a una distancia equidistante del centro de masas por lo que se cumple que $R_R = R_L = R_N = 0.5b$. En la práctica, se instalará un acelerómetro encima de cada rueda para obtener el valor de las aceleraciones.

Para un anillo delgado donde $I_G = M_G R^2$ se asigna $R_N = b/2 = R$ y se muestra la deducción de sus parámetros en la ecuación (B.27).

$$\begin{aligned} M_L = M_R &= \frac{I_G}{2R_N^2} = \frac{M_G R^2}{2R^2} = 0,5 M_G,\ \gamma = 0.5 \\ M_c &= M_G - \left(\frac{I_G}{R_N^2}\right) = M_G - \left(\frac{M_G R^2}{R^2}\right) = 0,\ \delta = 0 \\ \lambda_\alpha &= \frac{M_G R_N}{I_G} = \frac{M_G R}{M_G R^2} = \frac{1}{R} \\ \lambda_a &= \frac{\gamma}{(1-\delta)} = 0.5 \end{aligned} \tag{B.27}$$

Para un cilindro donde $I_G = 0.5 M_G R^2$ se realiza la misma asignación $R_N = b/2 = R$ y se muestra la deducción de sus parámetros en la ecuación (B.28).

$$\begin{aligned} M_L = M_R &= \frac{I_G}{2R_N^2} = \frac{0.5 M_G R^2}{2R^2} = 0,25 M_G,\ \gamma = 0,25 \\ M_c &= M_G - \left(\frac{I_G}{R_N^2}\right) = M_G - \left(\frac{0.5 M_G R^2}{R^2}\right) = 0,5 M_G,\ \delta = 0,5 \\ \lambda_\alpha &= \frac{M_G R_N}{I_G} = \frac{M_G R}{0.5 M_G R^2} = \frac{2}{R} \\ \lambda_a &= \frac{\gamma}{(1-\delta)} = \frac{0,25}{(1-0,5)} = 0.5 \end{aligned} \tag{B.28}$$

Por último, para un prisma donde $I_G = \frac{1}{12}\left(b^2+c^2\right)M_G$ se asigna de nuevo $R_N = 0.5b$ y se muestra la deducción de sus parámetros en la ecuación (B.29).

$$\begin{aligned}
M_L = M_R &= \frac{I_G}{2R_N^2} = \frac{(1/12)\,M_G\left(b^2+c^2\right)}{2(0.5d)^2} = \frac{M_G}{6}\left(1+(c/b)^2\right) \\
&\Rightarrow \gamma = \frac{1}{6}\left(1+(c/b)^2\right) \\
M_c &= M_G - \left(\frac{I_G}{R_N^2}\right) = M_G - \left(\frac{M_G}{3}\left(1+(c/b)^2\right)\right) = M_G\left(1-\frac{1+(c/b)^2}{3}\right) \\
&\Rightarrow M_c = \frac{M_G}{3}\left(2+(c/b)^2\right) \Rightarrow \delta = \left(1-\frac{1+(c/b)^2}{3}\right) = (1-2\gamma) \\
\lambda_\alpha &= \frac{M_G R_N}{I_G} = \frac{M_G\,(0.5d)}{(1/12)\,M_G b^2\left(1+(c/b)^2\right)} = \frac{6}{b\left(1+(c/b)^2\right)} = \frac{1}{b\gamma} \\
\lambda_a &= \frac{\gamma}{(1-\delta)} = \frac{\gamma}{(1-(1-2\gamma))} = 0.5
\end{aligned} \tag{B.29}$$

De nuevo, cabe destacar que el presente análisis se puede repetir para distintas plataformas de forma algebraica si se tiene la ecuación para el I_G o bien de forma numérica sustituyendo los valores correspondientes a M_G, I_G.

La tabla B.2 muestra un resumen de los parámetros del sistema de tres partículas aplicados a las formas genéricas de un robot en configuración diferencial.

Llegado este punto, se aplica la segunda ley de Newton al sistema de partículas para obtener el modelo dinámico (B.30).

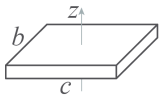
$$\begin{aligned}
\sum F &= F_R + F_L = M_L a_L + M_c a_c + M_R a_R \\
\sum \tau &= \tau_R - \tau_L = R_N\left(M_R a_R - M_L a_L\right)
\end{aligned} \tag{B.30}$$

Si se aplica el principio de D'Alembert al sistema dinámico (B.14) y al sistema dinámico por partículas (B.30) se obtienen las ecuaciones (B.31) y (B.32) respectivamente.

$$\begin{aligned}
\sum F &\Rightarrow M_G a - (F_R + F_L) = 0 \\
\sum \tau &\Rightarrow I_G \alpha - (\tau_R - \tau_L) = 0
\end{aligned} \tag{B.31}$$

$$\begin{aligned}
\sum F &\Rightarrow M_R a_R + M_c a_c + M_L a_L - (F_R + F_L) = 0 \\
\sum \tau &\Rightarrow R_N M_R a_R - R_N M_L a_L - (\tau_R - \tau_L) = 0
\end{aligned} \tag{B.32}$$

Tabla B.2: Parámetros del sistema de partículas dinámicamente equivalente, caso tres partículas para el robot diferencial

Forma del Robot	Momento de Inercia	γ	δ	λ_a	λ_α
Anillo Delgado 	$I_G = M_G R^2$	0.5	0	0.5	$\frac{1}{R}$
Cilindro Sólido 	$I_G = \frac{M_G R^2}{2}$	0.25	0.5	0.5	$\frac{2}{R}$
Rectángulo Sólido 	$I_G = \beta M_G$ $\beta = \frac{1}{12}\left(b^2+c^2\right)$	$\frac{1}{6}\left(1+(c/b)^2\right)$	$1-2\gamma$	0.5	$\frac{1}{b\gamma}$

Las cuales, al igualarlas entre sí dan como resultado la ecuación (B.33).

$$\begin{aligned} &M_G a = M_R a_R + M_c a_c + M_L a_L \\ &I_G \alpha = R_N M_R a_R - R_N M_L a_L \Leftrightarrow R_R = R_L = R_N = 0.5b \end{aligned} \tag{B.33}$$

En la ecuación (B.33) se observa que el cuerpo rígido (B.14) y el sistema de partículas (B.30) son dinámicamente equivalentes al asignar $R_R = R_L = R_N = 0.5b$ y al utilizar los parámetros de la ecuación (B.26), definidos según la forma del robot de acuerdo con la tabla B.2. Tal y como se ha mencionado anteriormente, en la práctica se colocarán los acelerómetros sobre las ruedas del robot para medir (a_R, a_L).

Con estas condiciones se resuelve la ecuación (B.33) para las aceleraciones locales a y α (con $a_c = a$) al sustituir (B.26) en (B.33), tal y como se muestra en la ecuación (B.34)

$$\begin{aligned} M_G a &= M_R a_R + M_c a_c + M_L a_L \Rightarrow M_G a - \delta M_G a = \gamma M_G \left(a_R + a_L\right) \\ &\Rightarrow a M_G = \frac{\gamma}{(1-\delta)} M_G \left(a_R + a_L\right) \Rightarrow a = \lambda_a \left(a_R + a_L\right) \\ I_G \alpha &= M_G R_N \gamma \left(a_R - a_L\right) \\ &\Rightarrow \alpha = \frac{M_G R_N}{I_G} \gamma \left(a_R - a_L\right) \Rightarrow \alpha = \lambda_\alpha \gamma \left(a_R - a_L\right) \\ \therefore a &= \lambda_a \left(a_R + a_L\right),\ \alpha = \lambda_\alpha \gamma \left(a_R - a_L\right) \end{aligned} \tag{B.34}$$

Estas aceleraciones sirven como entrada a los modelos (B.23) y (B.24) anteriormente descritos, los cuales son utilizados en el proceso de estimación de la pose del robot. Estos modelos de tres partículas se utilizarán para los filtros de localización cooperativa entre grupos de robots, ya que son más generales que sus versiones de dos partículas, y son aplicables a las distintas plataformas que conforman un grupo de robots heterogéneos.

Una vez descritos los modelos utilizados para robots de configuración diferencial, se procede con la obtención de los modelos para un robot en configuración Ackerman, utilizado también para las demostraciones prácticas de la presente tesis.

B.2 Configuración Ackerman

Por definición, un robot en configuración Ackerman consiste en un cuerpo rígido con centro de masa y gravedad P_0, radio de rotación R_G, masa M_G y un momento de inercia I_G con dos ruedas traseras no deformables y no orientables (fijas) separadas una distancia b entre ellas y a una distancia l de dos ruedas delanteras orientables (también a una distancia b entre ellas) tal y como se muestra en la figura B.5.

La dirección Ackerman modifica la orientación de las dos ruedas delanteras de forma que a baja velocidad todas las ruedas cumplen con la condición de rodamiento puro sin deslizamiento lateral. Tal y como se demuestra en [194], esto ocurre debido a que a bajas velocidades cada rueda sigue una trayectoria curva con radios distintos pero con un centro instantáneo de rotanción común C_r (Fig. B.5(a)). En los trabajos de [194] y [142], esta configuración se plantea como un modelo equivalente de bicicleta suponiendo que el movimiento se restringe a un único plano horizontal (Fig. B.5(b)). De este modo, para las ruedas delanteras se utiliza el promedio ϕ de sus ángulos (ϕ_i, ϕ_o), reduciéndolas a una única rueda; y de igual manera, las traseras se consideran como otra única rueda equivalente donde se aplica la fuerza del motor.

En [142] se describen las condiciones necesarias para que no exista deslizamiento a bajas velocidades, sin embargo en los experimentos prácticos de este trabajo, no ha sido posible corroborar el cumplimiento de dichas condiciones por lo que para la obtención del modelo se ha considerado que puedan ocurrir deslizamientos.

B.2.1 Modelo cinemático

El modelo cinemático Ackeman representa la evolución de las velocidades del robot en un marco inercial fijo. La *pose* o *postura* del robot se define mediante las coordenadas del punto $P_0 = (x, y)$ y el ángulo de avance θ en el marco de referencia *global* (X_G,Y_G) en la figura B.5. Conociendo las velocidades lineales y angulares (v y ω) en el marco de referencia local (X_L,Y_L),la velocidad *global* del robot se define según la ecuación (B.35).

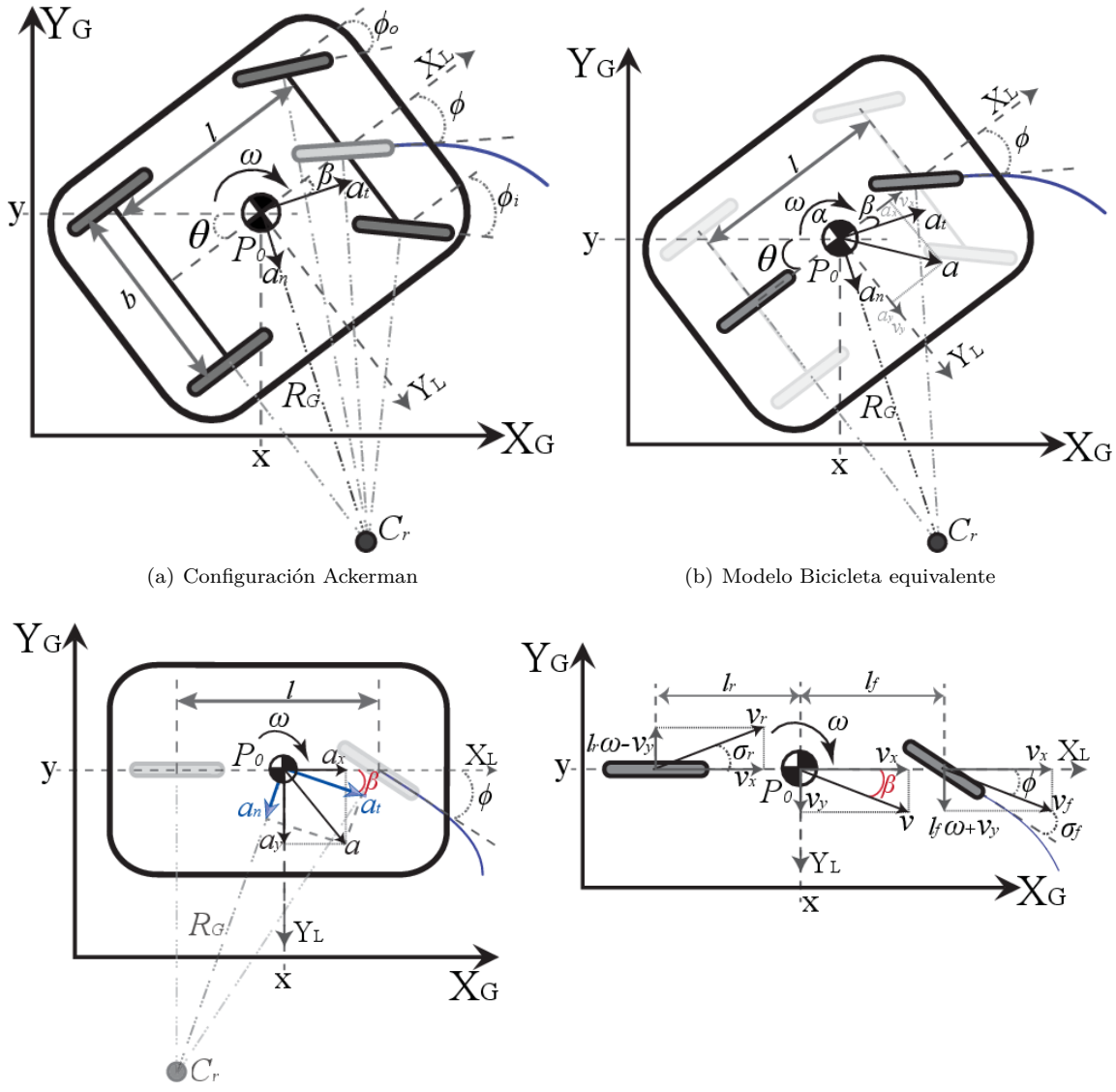
$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{B.35}$$

Discretizando e integrando recursivamente en (B.35) con el tiempo de muestreo T_s, se obtiene la *pose global* del robot $\boldsymbol{L}$ en (B.36).

$$\boldsymbol{L}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{k-1} + T_s \begin{bmatrix} \cos\theta_{k-1} & -\sin\theta_{k-1} & 0 \\ \sin\theta_{k-1} & \cos\theta_{k-1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}_{k-1} \tag{B.36}$$

Cuando el robot realiza una trayectoria circular, la posición del robot sufre las transformaciones de traslación y rotación simultáneamente. Mediante el procedimiento descrito en [194], de la aceleración a se pueden obtener sus componentes a_x, a_y del centro de gravedad P_0 utilizando las aceleraciones normal y tangencial (a_n,a_t), mostradas en la figura B.5(c). El componente tangencial a_t tiene la misma dirección que la velocidad resultante v del centro de gravedad, la cual se encuentra a un ángulo β del eje de avance X_L. Este ángulo es conocido como el ángulo de deslizamiento lateral del vehículo. Las aceleraciones tangencial a_t y normal a_n se descomponen según los ejes locales tal y como se muestra en la ecuación (B.37).

$$\begin{aligned} a_t &= \begin{cases} \dot{v}_x \text{ Eje } \mathrm{X}_L \\ \dot{v}_y \text{ Eje } \mathrm{Y}_L \end{cases} \\ a_n &= \begin{cases} -\left(v^2/R\right)\sin\beta = -v\omega \cdot \sin\beta = -v_y\omega \text{ Eje } \mathrm{X}_L \\ \left(v^2/R\right)\cos\beta = v\omega \cdot \cos\beta = v_x\omega \text{ Eje } \mathrm{Y}_L \end{cases} \end{aligned} \tag{B.37}$$



(a) Configuración Ackerman

(b) Modelo Bicicleta equivalente

(c) Cinemática del Modelo de la Bicicleta equivalente [194]

Figura B.5: Cinemática de un robot en configuración Ackerman

Al agrupar en los ejes locales (X_L,Y_L) los términos de las aceleraciones a_n y a_t, tal y como se especifica en [194], se obtienen los componentes a_x, a_y de la aceleración a.

$$\begin{aligned} a_x &= \dot{v_x} - v_y\omega \\ a_y &= \dot{v_y} + v_x\omega \end{aligned} \tag{B.38}$$

Para finalizar el modelo cinemático del modelo, la ecuación (B.39) define la relación cinemática entre el ángulo de la dirección Ackerman ϕ con el ángulo de avance del robot ω.

$$\omega = (v_x \tan\phi)/l \tag{B.39}$$

B.2.2 Modelo dinámico

Este modelo representa la evolución de las aceleración linear y angular del robot (a_x,a_y,α) en términos de las fuerzas que han sido aplicadas por los motores en las ruedas frontal F_f y trasera F_r y el par angular τ. De forma similar al caso diferencial, los modelos en la literatura ([188, 200, 51, 2] y [130]) describen la dinámica en los ejes globales, por lo que nuevamente se requiere aplicar el análisis por partículas dinámicamente equivalentes para obtener los modelos a utilizar en los filtros de fusión.

Para este caso, se expone directamente el modelo de sistemas de tres partículas dinámicamente equivalentes para la obtención del modelo dinámico.

B.2.2.1 Sistema de tres partículas dinámicamente equivalente

El cuerpo rígido de un robot en configuración Ackerman de masa M_G y momento de inercia I_G, se estudia como un sistema dinámicamente equivalente formado por tres partículas con masa M_r, M_c y M_f unidas por dos barras (conectores) sin masa, de longitud constante R_r y R_f tal y como se muestra en la figura B.6.

Nuevamente,se deben aplicar las condiciones expuestas en [14]:

- Conservación de la masa: $M_G = M_f + M_c + M_r$
- Conservación del centro de masa: $M_f R_f = M_r R_r$
- Conservación del momento de inercia: $M_f R_f^2 + M_r R_r^2 = I_G$

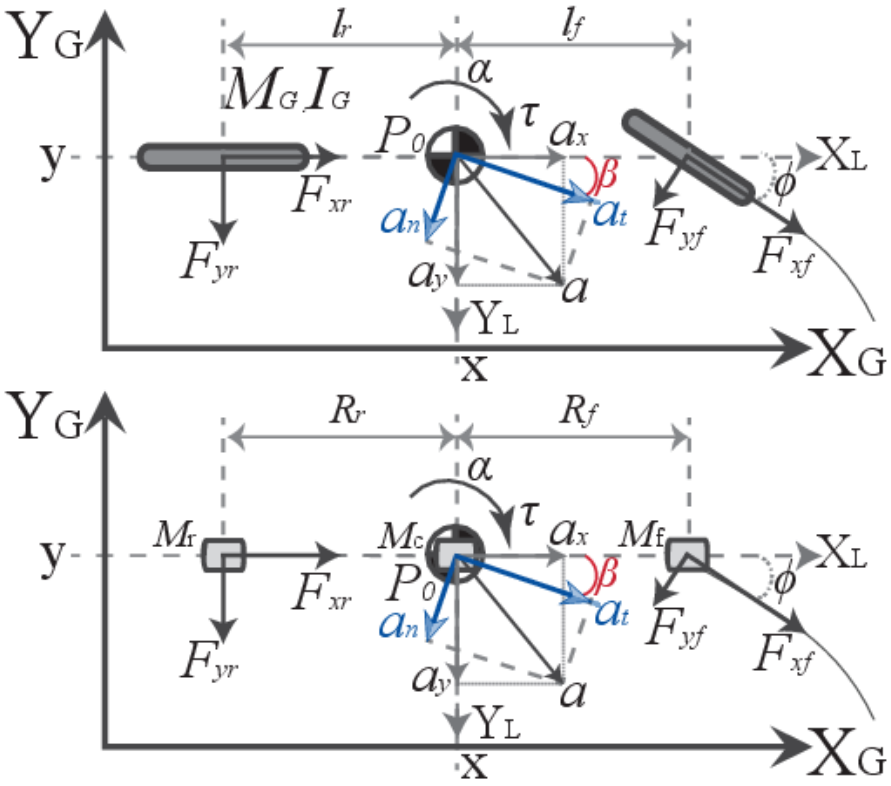


Figura B.6: Modelo dinámico de un robot en configuración Ackerman como un cuerpo rígido y como un sistema de tres partículas equivalente

De este modo se despejan las masas del sistema equivalente:

$$M_f = \frac{I_G}{R_f\left(R_f + R_r\right)}, M_r = \frac{I_G}{R_r\left(R_f + R_r\right)}, M_c = M_G - \left(\frac{I_G}{R_f R_r}\right) \tag{B.40}$$

Ya que las distancias entre partículas son independientes, por simplicidad se escoge que sean iguales entre sí ($R_f = R_r = R_N$). De este modo, es posible reducir la ecuación (B.40) a (B.41), donde aprovechando que las masas M_f y M_r son proporcionales a la masa total M_G, dicha relación se ha expresado mediante la constante γ ($0 \leq \gamma < 1$). De igual manera, M_c es proporcional a M_G y se ha expresado con la constante δ ($0 \leq \delta < 1$). Adicionalmente se han definido dos constantes más, λ_a utilizando γ y δ, y λ_α mediante M_G, R_N e I_G.

$$\begin{array}{ll} M_f = M_r = \frac{I_G}{2R_N^2} = \gamma M_G, & \lambda_a = \frac{\gamma}{1-\delta} \\ M_c = M_G - \left(\frac{I_G}{R_N^2}\right) = \delta M_G, & \lambda_\alpha = \frac{M_G R_N}{I_G} \end{array} \tag{B.41}$$

En cuanto a la forma física del robot, en este caso se ha obtado por aplicar una forma rectangular de longitud c y ancho b, lo cual responde a la morfología mayoritaria de este tipo de vehículos; por lo que $R_f = l_f = R_r = l_r = 0.5l$ según la figura B.6. De esta forma se sustituye $I_G = \frac{1}{12}\left(b^2 + c^2\right) M_G$ en (B.41) asignando $R_N = 0.5l = 0.5c$ y se obtienen los parámetros de la ecuación (B.42).

$$\begin{array}{cc} I_G = \frac{1}{12} M_G\left(b^2 + c^2\right), & \gamma = \frac{1}{6}\left(1 + (b/c)^2\right) \\ \delta = \left(1 - \frac{1+(b/c)^2}{3}\right), & \lambda_\alpha = \frac{6}{c\left(1+(b/c)^2\right)} \\ \lambda_a = 0.5, & \lambda_\alpha \gamma = 1/c \end{array} \tag{B.42}$$

Una vez obtenidos dichos parámetros, queda aplicar la segunda Ley de Newton al cuerpo rígido y al sistema de partículas equivalente y se obtienen (B.43) y (B.44).

$$\begin{array}{l} \Sigma F_x = \left(F_{xf}\cos\phi - F_{yf}\sin\phi\right) + F_{xr} = M_G a_x \\ \Sigma F_y = \left(F_{yf}\cos\phi + F_{xf}\sin\phi\right) + F_{yr} = M_G a_y \\ \Sigma\tau = l_f\left(F_{yf}\cos\phi + F_{xf}\sin\phi\right) - l_r F_{yr} = I_G\alpha \end{array} \tag{B.43}$$

$$\begin{array}{l} \Sigma F_x = \left(F_{xf}\cos\phi - F_{yf}\sin\phi\right) + F_{xr} = M_f a_{f,x} + M_c a_{c,x} + M_r a_{r,x} \\ \Sigma F_y = \left(F_{yf}\cos\phi + F_{xf}\sin\phi\right) + F_{yr} = M_f a_{f,y} + M_c a_{c,y} + M_r a_{r,y} \\ \Sigma\tau = R_f\left(F_{yf}\cos\phi + F_{xf}\sin\phi\right) - R_r F_{yr} = R_f M_f a_{f,y} - R_r M_r a_{r,y} \end{array} \tag{B.44}$$

Aplicando el principio de D'Alembert a (B.43) y (B.44) se obtienen (B.45) y (B.46) respectivamente.

$$\begin{aligned} &M_G a_x - (F_{xf}\cos\phi - F_{yf}\sin\phi + F_{xr}) = 0 \\ &M_G a_y - (F_{yf}\cos\phi + F_{xf}\sin\phi + F_{yr}) = 0 \\ &I_G\alpha - (l_f F_{yf}\cos\phi + l_f F_{xf}\sin\phi - l_r F_{yr}) = 0 \end{aligned} \tag{B.45}$$

$$\begin{aligned} &M_f a_{f,x} + M_c a_{c,x} + M_r a_{r,x} - (F_{xf}\cos\phi - F_{yf}\sin\phi + F_{xr}) = 0 \\ &M_f a_{f,y} + M_c a_{c,y} + M_r a_{r,y} - (F_{yf}\cos\phi + F_{xf}\sin\phi + F_{yr}) = 0 \\ &(R_f M_f a_{f,y} - R_r M_r a_{r,y}) - (R_f F_{yf}\cos\phi + R_f F_{xf}\sin\phi - R_r F_{yr}) = 0 \end{aligned} \tag{B.46}$$

Al igualar (B.45) y (B.46) se obtiene (B.47).

$$\begin{aligned} &M_G a_x = M_f a_{f,x} + M_c a_{c,x} + M_r a_{r,x} \\ &M_G a_y = M_f a_{f,y} + M_c a_{c,y} + M_r a_{r,y} \\ &I_G\alpha = (R_f M_f a_{f,y} - R_r M_r a_{r,y}) \Leftrightarrow R_f = l_f, R_r = l_r \end{aligned} \tag{B.47}$$

Como se puede observar, el cuerpo rígido y el sistema de tres partículas son dinámicamente equivalentes al utilizar los parámetros de la ecuación (B.42) cuando $R_f = l_f, R_r = l_r$. Como se mencionó anteriormente, se utilizará $R_f = l_f = R_r = l_r = 0.5l$ para el sistema equivalente del robot Ackerman, siendo esta la distancia a la que físicamente se colocarán los acelerómetros en el centro de los ejes de la plataforma experimental. Utilizando estas condiciones se resuelve para las aceleraciones locales a_x, a_y y α (con $a_c = a$) tal y como se muestra en la ecuación (B.48). Éstas resultan ser similares al caso diferencial (ecuación (B.34)) pero con deslizamiento.

$$\begin{aligned} &M_G a_x – \delta M_G a_x = \gamma M_G\,(a_{f,x} + a_{r,x}) \Rightarrow a_x = \frac{\gamma}{1–\delta}\,(a_{f,x} + a_{r,x}) = \lambda_a\,(a_{f,x} + a_{r,x}) \\ &M_G a_y – \delta M_G a_y = \gamma M_G\,(a_{f,y} + a_{r,y}) \Rightarrow a_y = \frac{\gamma}{1–\delta}\,(a_{f,y} + a_{r,y}) = \lambda_a\,(a_{f,y} + a_{r,y}) \\ &\alpha = \frac{\gamma M_G}{I_G}\,(R_f a_{f,y} – R_r a_{r,y}) = \gamma\frac{M_G}{I_G} R_N\,(a_{f,y} – a_{r,y}) = \lambda_\alpha\gamma\,(a_{f,y} – a_{r,y}) \\ &\therefore a_x = \lambda_a\,(a_{f,x} + a_{r,x}),\ a_y = \lambda_a\,(a_{f,y} + a_{r,y}),\ \alpha = \lambda_\alpha\gamma\,(a_{f,y} - a_{r,y}) \end{aligned} \tag{B.48}$$

Las aceleraciones se sustituyen en (B.38) para tomar en cuenta los componentes de la aceleración normal y se obtiene:

$$\begin{aligned}\dot{v}_x &= v_y\omega + \lambda_a\left(a_{f,x} + a_{r,x}\right) = v_y\omega + \lambda_a u_1\\ \dot{v}_y &= -v_x\omega + \lambda_a\left(a_{f,y} + a_{r,y}\right) = -v_x\omega + \lambda_a u_2\\ \dot{\omega} &= \lambda_\alpha\gamma\left(a_{f,y} - a_{r,y}\right) = \lambda_\alpha\gamma u_3\end{aligned} \tag{B.49}$$

La ecuación (B.49) corresponde al modelo dinámico local en el que se han definido las entradas u_1, u_2, u_3 como los respectivos términos de la aceleraciones de entrada en las ecuaciones de $v_x, v_y, \dot{\omega}$ para facilitar la notación. Se observa que el modelo (B.49) del robot Ackerman con deslizamiento, corresponde a un modelo no lineal, con lo que resulta conveniente aproximar la solución de la integral discreta mediante el método numérico de Euler, pudiendo utilizarse otros métodos numéricos o resolver la integral exacta (siguiendo un procedimiento similar a la cinemática diferencial) si se desea más precisión o se disponen de más recursos en la plataforma. De esta forma, obteniendo la discretización mediante el método de Euler, el modelo dinámico del robot se obtiene en (B.50).

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}_k = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}_{k-1} + T_s \begin{bmatrix} v_y\omega + \lambda_a u_1 \\ -v_x\omega + \lambda_a u_2 \\ \lambda_\alpha\gamma u_3 \end{bmatrix}_{k-1} \tag{B.50}$$

Sustituyendo (B.50) como entradas de (B.36) se obtiene el modelo dinámico *global* del robot:

$$\begin{bmatrix} x \\ y \\ \theta \\ v_x \\ v_y \\ \omega \end{bmatrix}_k = \begin{bmatrix} x \\ y \\ \theta \\ v_x \\ v_y \\ \omega \end{bmatrix}_{k-1} + T_s \begin{bmatrix} v_x\cos\theta - v_y\sin\theta \\ v_x\sin\theta + v_y\cos\theta \\ \omega \\ v_y\omega + \lambda_a u_1 \\ -v_x\omega + \lambda_a u_2 \\ \lambda_\alpha\gamma u_3 \end{bmatrix}_{k-1} \tag{B.51}$$

Para terminar con el modelo dinámico, en caso de que el robot viaje a baja velocidad, se puede aprovechar la condición de no deslizamiento $v_y \approx 0$ ([142], [194]) para obtener los modelos local y global simplificados, tal y como se muestra en (B.52) y (B.53).

$$\begin{bmatrix} v_x \\ \omega \end{bmatrix}_k = \begin{bmatrix} v_x \\ \omega \end{bmatrix}_{k-1} + T_s \begin{bmatrix} \lambda_a u_1 \\ \lambda_\alpha\gamma u_3 \end{bmatrix}_{k-1} \tag{B.52}$$

$$\begin{bmatrix} x \\ y \\ \theta \\ v_x \\ \omega \end{bmatrix}_k = \begin{bmatrix} x \\ y \\ \theta \\ v_x \\ \omega \end{bmatrix}_{k-1} + T_s \begin{bmatrix} v_x \cos\theta \\ v_x \sin\theta \\ \omega \\ \lambda_a u_1 \\ \lambda_\alpha \gamma u_3 \end{bmatrix}_{k-1} \tag{B.53}$$

Finalmente, los modelos (B.36), (B.48) y de (B.50) a (B.53) han sido los utilizados con el método de fusión sensorial para estimar la *pose* del robot, tal y como se describe en la sección 3.2.

B.3 Modelo dinámico de los motores

En caso de no disponer de acelerómetros en la plataforma, se puede identificar un modelo para las velocidades de las ruedas del motor y calcular su derivada para obtener una aproximación de la aceleración lineal o angular de cada rueda. Estos modelos pueden ser utilizados como entrada a los modelos basados en el sistema de partículas ((B.22),(B.34) y (B.52)) los cuales a su vez funcionan como entrada del modelo dinámico local (B.23) o global (B.24) para el robot diferencial o en el modelo del caso sin deslizamiento del robot Ackerman (B.53). Este tipo de modelos son comunes y suelen obtenerse para realizar el diseño del controlador PID de regulación de velocidad de los motores. En general se utilizan una serie de escalones positivos y negativos en la acción de control con el fin de obtener la función de transferencia de bucle abierto del motor del robot. A manera de ejemplo se utilizará el modelo de velocidad de la ecuación (B.54) el cual es obtenido para los motores DC de un robot diferencial construido utilizando la plataforma LEGO Mindstorms NXT, tal y como se detalla en el apéndice C.

$$G_{gMA} = \frac{0.1276}{0.1235\ s + 1} \tag{B.54}$$

Para determinar la aceleración de las ruedas se deriva la ecuación (B.54) para obtener (B.55).

$$G_{gAcc} = \frac{a_{ccRueda}}{V_{control}} = \frac{0.1276\ s}{0.1235\ s + 1} \tag{B.55}$$

Este modelo relaciona la acción de control $V_{control}$ con la aceleración de la rueda $a_{ccRueda}$. Se discretiza (B.55) con un periodo de muestreo $T_s = 50ms$ con lo que se obtiene (B.56). Ésta se escribe en forma de ecuación en diferencias en (B.57) lo que facilita su implementación dentro del robot.

$$G_{gAcc}(z) = \frac{1.033z - 1.033}{z - 0.6671} = \frac{1.033 - 1.033z^{-1}}{1 - 0.6671z^{-1}} \tag{B.56}$$

$$a_{R,k} = 1.033V_{c,k} - 1.033V_{c,k-1} + 0.6671a_{R,k-1} \tag{B.57}$$

Apéndice C

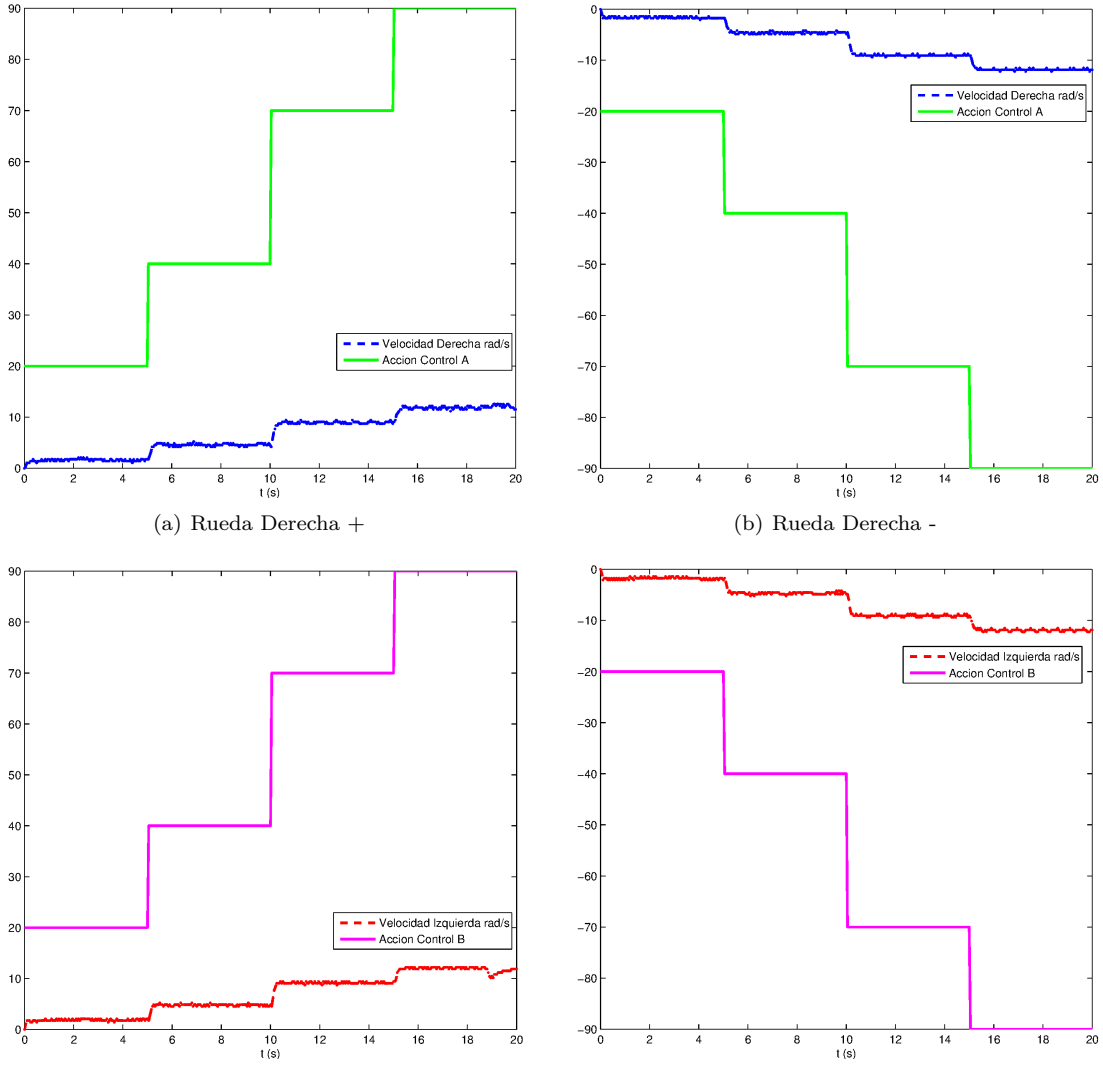
Modelo dinámico de los motores del LEGO Mindtorms NXT

Se identifica a continuación un modelo para las velocidades de las ruedas del motor y calcular su derivada para obtener una aproximación a la aceleración lineal o angular de cada rueda. Este tipo de modelos pueden ser utilizados como entrada a los modelos de aceleraciones propuestos en el anexo B en caso de no disponer de acelerómetros en la plataforma.

En lugar de utilizar un único escalón para realizar la identificación, se utiliza una seria de escalones positivos y negativos en la acción de control con el fin de obtener la función de transferencia de bucle abierto del motor del robot. El procedimiento expuesto a continuación sigue el ejemplo de identificación en un robot diferencial con motores DC basado en la plataforma LEGO Mindtorms NXT, pudiendo ser aplicado en otras plataformas comerciales o de construcción propia.

La identificación del modelo de velocidad angular de los motores del robot se realiza al introducir un escalón en la acción de control, para obtener la respuesta dinámica del motor como una función de transferencia de bucle abierto de primer orden para la velocidad. En el caso del LEGO NXT, la acción de control es el voltaje PWM especificado en %, así un valor de +100 indica que se aplica el voltaje máximo posible al motor en sentido horario. Se realizan dos pruebas con escalones positivos y negativos para cada motor (cuatro pruebas en total), las cuales se observan en la figura C.1.

En total se deben identificar 8 funciones de transferencia para cada motor (16 en total). Para facilitar la tarea se utiliza el «System Identification Toolbox» de Matlab ®(pueden emplearse herramientas similares de identificación de modelos como el Scilab, Octave, entre otros). Primeramente se separan las pruebas completas en



(a) Rueda Derecha +

(b) Rueda Derecha -

(c) Rueda Izquierda +

(d) Rueda Izquierda -

Figura C.1: Pruebas realizadas a los motores del LEGO NXT

cuatro secciones cada una (una por escalón) y se identifica un modelo de primer orden sin tiempo muerto para cada una de éstas (modelos locales). Los parámetros del modelo se muestran en la tabla C.1 en donde se calculan los modelos con los parámetros promedio de cada rueda.

Tabla C.1: Modelos locales para los motores del LEGO NXT

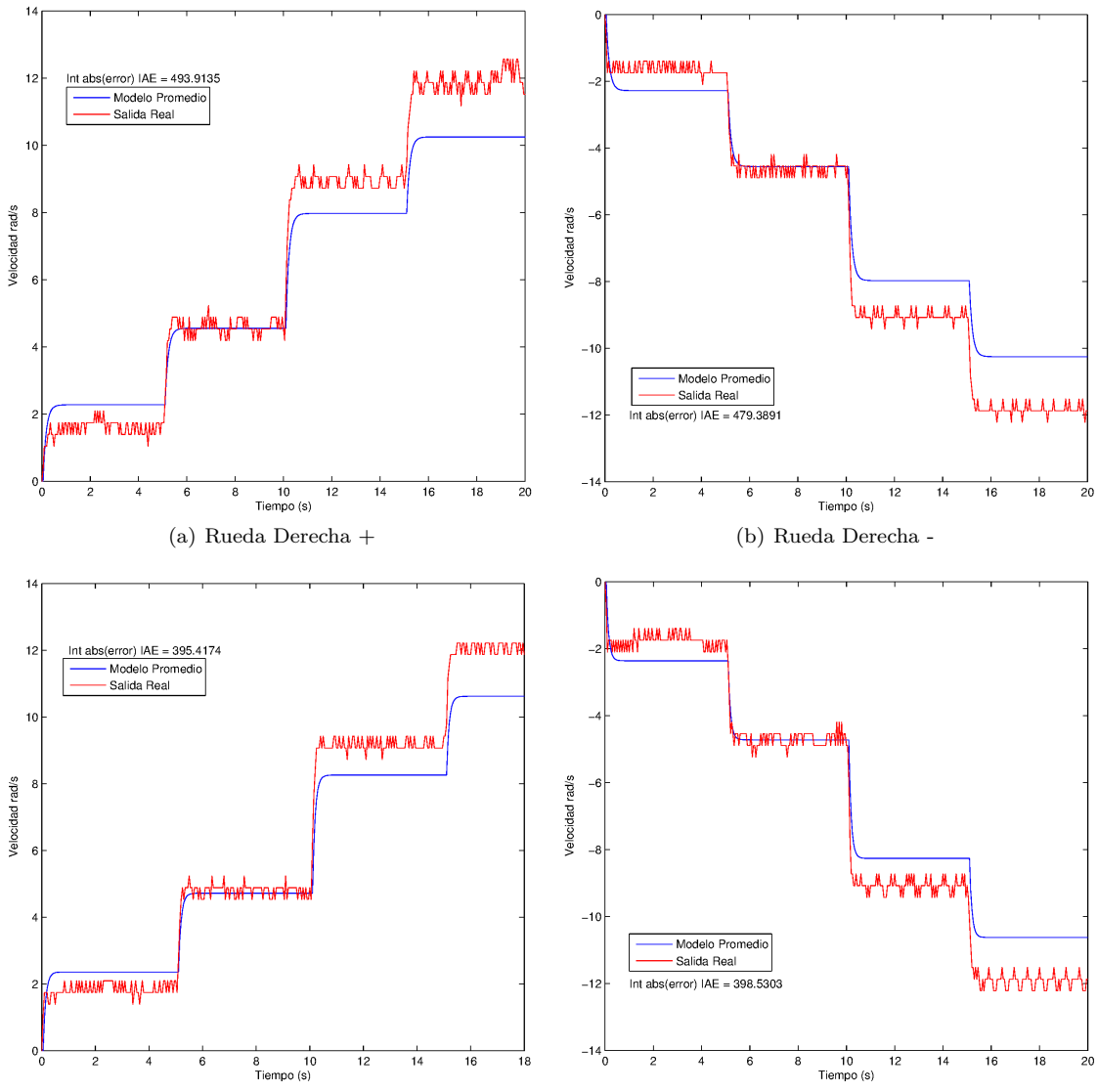
Escalón de Entrada	**Motor Rueda Derecha (A)**		**Motor Rueda Izquierda (B)**	
	Ganancia K_p	**Cte. Tiempo** τ	***Ganancia*** K_p	**Cte. Tiempo** τ
20	0.0809	0.1561	0.0918	0.0534
40	0.1146	0.1452	0.1203	0.1062
70	0.1275	0.1342	0.1308	0.1003
90	0.1311	0.1359	0.1339	0.1080
-20	*0.0809*	0.0530	*0.0876*	0.0481
-40	*0.1158*	0.1567	*0.1177*	0.1374
-70	*0.1290*	0.1333	*0.1296*	0.1044
-90	*0.1314*	0.1120	*0.1324*	0.1077
PROMEDIO para	**0.1139**	**0.1283**	**0.1180**	**0.0957**
cada rueda (Local)	$G_{pMA} = \frac{0.1139}{0.1283\ s+1}$		$G_{pMB} = \frac{0.118}{0.0957\ s+1}$	

Para observar el comportamiento del modelo promedio (local/individual) se grafica su respuesta ante la entrada de prueba (serie de escalones) y se observa su desempeño al compararlo con la salida del motor real. Para esto se calcula la integral del valor absoluto del error (IAE) cuyo valor indica la cercanía de la estimación del modelo a la salida real (entre menor el IAE mejor la estimación). Las pruebas para el modelo se muestran en la figura C.2.

Se observa que los modelos promedio no aproximan de forma adecuada a todas las pruebas realizadas, ya que sus salidas difieren de forma considerable en velocidades mayores a 5 rad/s. Para mejorar la estimación se identifica un modelo global para cada rueda utilizando todas las entradas a la hora de hacer la identificación (pruebas para todos los escalones positivos y negativos en conjunto, no por separado, utilizando por ejemplo la función «pem» en Matlab). Los resultados de la identificación se resumen en la tabla C.2 en donde se calculan los modelos promedio de cada rueda a partir de los modelos globales.

Tabla C.2: Modelos globales para los motores del LEGO NXT

Escalón de Entrada	**Motor Rueda Derecha (A)**		**Motor Rueda Izquierda (B)**	
	Ganancia K_p	**Cte. Tiempo** τ	***Ganancia*** K_p	**Cte. Tiempo** τ
20, 40, 70, 90	0.1261	0.1513	0.1297	0.1062
-20, -40, -70, -90	0.1269	0.1307	0.1279	0.1059
PROMEDIO para	**0.1265**	**0.1410**	**0.1288**	**0.1061**
cada rueda (Global)	$G_{gMA} = \frac{0.1265}{0.141\ s+1}$		$G_{gMB} = \frac{0.1288}{0.1061\ s+1}$	



(a) Rueda Derecha +

(b) Rueda Derecha -

(c) Rueda Izquierda +

(d) Rueda Izquierda -

Figura C.2: Respuesta del modelo local promedio de las ruedas del LEGO NXT

Se observa que ambos modelos son muy similares entre sí (entre motores y entre pruebas). Para observar el comportamiento del modelo promedio (global de cada rueda) se grafica su respuesta ante la entrada de prueba (serie de escalones) y se observa su desempeño al compararlo con la salida del motor real. Nuevamente se calcula el valor de IAE para poder comparar con los modelos promedio locales. Las pruebas para el modelo se muestran en la figura C.3.

Se observa que el modelo promedio global de cada rueda se desempeña mejor que el promedio local ya que el valor de IAE es cercano a la mitad que en el caso local.

Para simplificar aún más el diseño del controlador se obtiene el modelo global promedio calculando el promedio de los modelos de cada rueda. Esto se muestra en la ecuación (C.1). Su desempeño en las pruebas con los datos de los motores se muestra en la figura C.4.

$$G_{gMA} = \frac{0.1276}{0.1235\ s + 1} \tag{C.1}$$

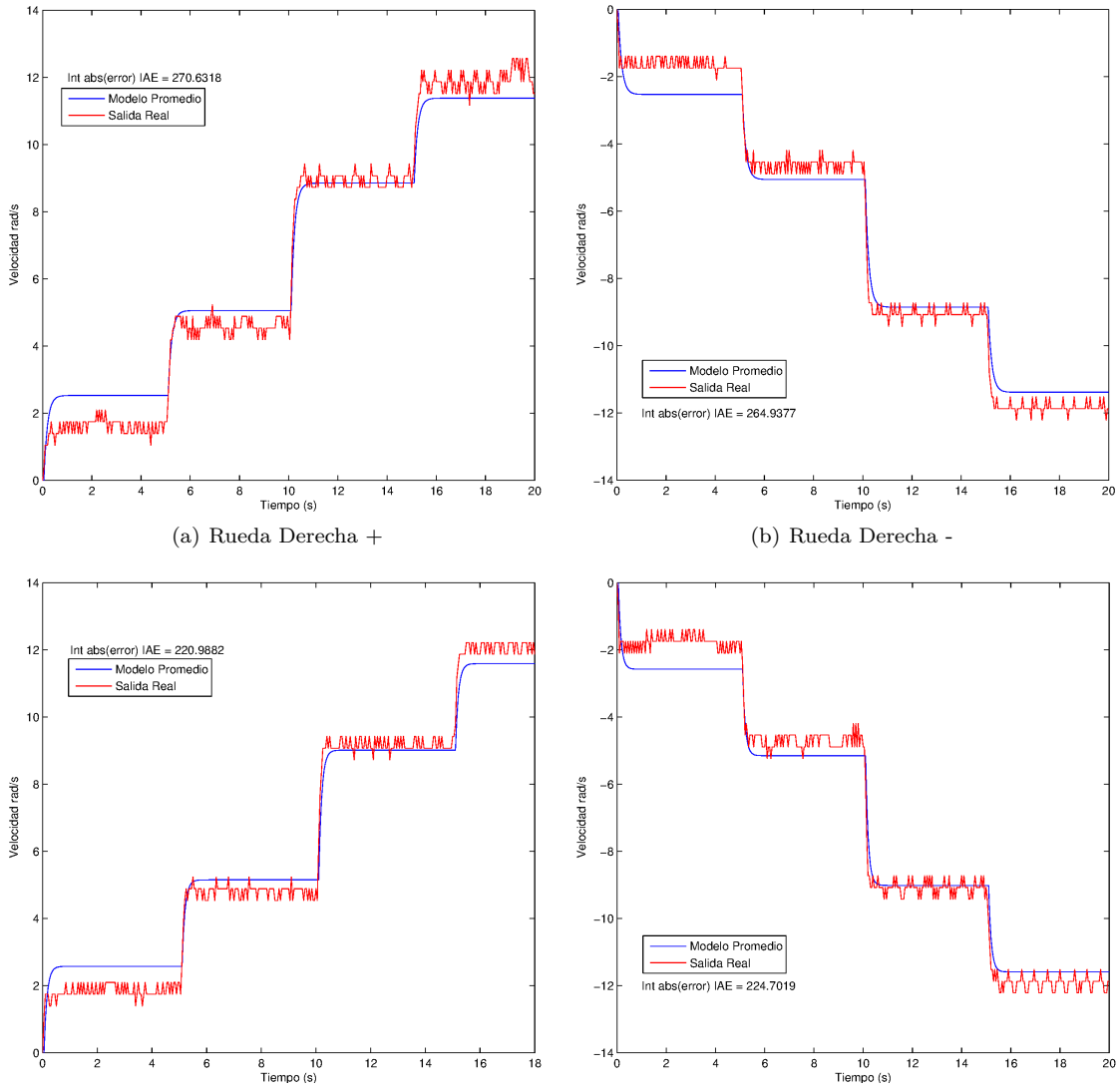
Se observa que el modelo global para ambas ruedas funciona muy similar a los modelos globales para cada rueda por separado, se mejora un poco el IAE en la rueda derecha y empeora en la izquierda, sin embargo esta variación es pequeña por lo que se considera aceptable con el fin de realizar un único diseño del control para las ruedas de los motores. Además, cabe señalar que al utilizar un tren de escalones positivo y negativo junto con la identificación global para obtener el modelo, se consigue un menor índice IAE que en el caso local (al identificar cada modelo por separado) y en el caso de emplear el modelo de un único escalón para todo el rango de actuación. Para determinar la aceleración de las ruedas se deriva la ecuación (C.1) para obtener (C.2).

$$G_{gAcc} = \frac{a_{ccRueda}}{V_{control}} = \frac{0.1276\ s}{0.1235\ s + 1} \tag{C.2}$$

Este modelo relaciona la acción de control $V_{control}$ con la aceleración de la rueda $a_{ccRueda}$. Se discretiza (C.2) con un periodo de muestreo $T_s = 50ms$ (adecuado para el LEGO NXT, puede modificarse según las características de la plataforma utilizada) con lo que se obtiene (C.3). Finalmente se escribe este modelo en forma de ecuación en diferencias en (C.4) lo que facilita su implementación dentro del robot.

$$G_{gAcc}(z) = \frac{1.033z - 1.033}{z - 0.6671} = \frac{1.033 - 1.033z^{-1}}{1 - 0.6671z^{-1}} \tag{C.3}$$

$$a_{R,k} = 1.033V_{c,k} - 1.033V_{c,k-1} + 0.6671a_{R,k-1} \tag{C.4}$$



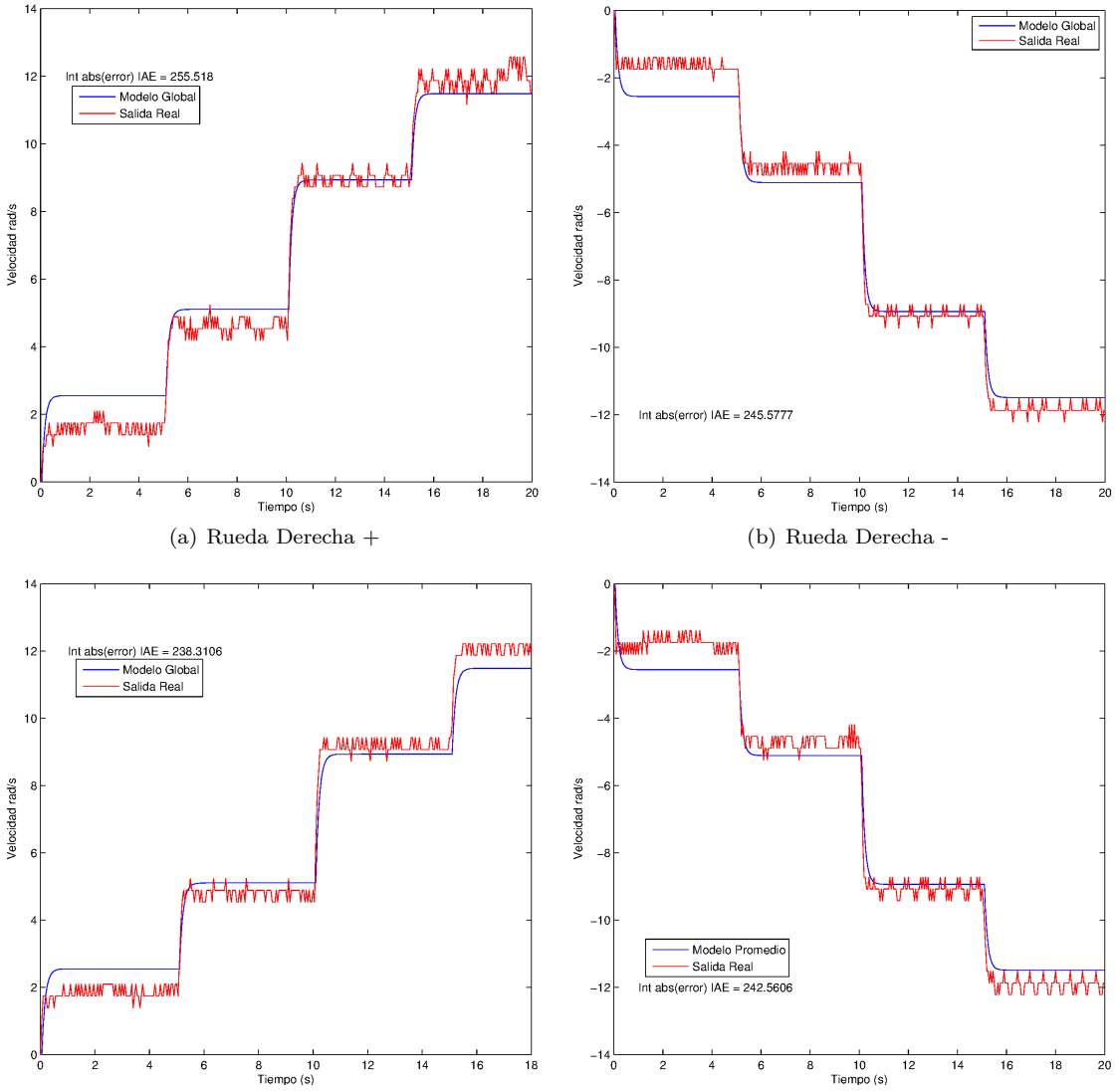
(a) Rueda Derecha +

(b) Rueda Derecha -

(c) Rueda Izquierda +

(d) Rueda Izquierda -

Figura C.3: Respuesta del modelo global de las ruedas del LEGO NXT



(a) Rueda Derecha +

(b) Rueda Derecha -

(c) Rueda Izquierda +

(d) Rueda Izquierda -

Figura C.4: Respuesta del modelo global Promedio de las ruedas del LEGO NXT

Bibliografía

[1] V. Botti A. Giret. *Holons and Agents*. Journal of Intelligent Manufacturing vol. 15, no. 5 pp. 645-659. 2004 (vid. pág. 16).

[2] Abdulgani Albagul, Wahyudi Martono y Riza Muhida. «Dynamic modelling and adaptive traction control for mobile robots». En: *Cutting Edge Robotics* (2005), pág. 3 (vid. págs. 289, 304).

[3] Javier Alonso-Mora y col. «Optimal reciprocal collision avoidance for multiple non-holonomic robots». En: *Distributed Autonomous Robotic Systems*. Springer, 2013, págs. 203-216 (vid. pág. 234).

[4] Yaniv Altshuler y col. «Multi-agent cooperative cleaning of expanding domains». En: *The International Journal of Robotics Research* 30.8 (2011), págs. 1037-1071 (vid. pág. 154).

[5] Francesco Amigoni, Monica Reggiani y Viola Schiaffonati. «An insightful comparison between experiments in mobile robotics and in science». En: *Autonomous Robots* 27.4 (2009), págs. 313-325 (vid. pág. 55).

[6] Noriaki Ando, Takashi Suehiro y Tetsuo Kotoku. «A software platform for component based rt-system development: Openrtm-aist». En: *Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2008, págs. 87-98 (vid. pág. 14).

[7] G.W. Andrews y col. *Communications system using a central controller to control at least one network and agent system*. US Patent 5,546,452. 1996 (vid. pág. 11).

[8] Leopoldo Armesto Ángel y Josep Tornero Montserrat. *Técnicas de control y fusión sensorial multifrecuenciales y su aplicación a la robótica móvil*. Universidad Politécnica de Valencia, 2005 (vid. pág. 127).

[9] Gianluca Antonelli, Stefano Chiaverini y Giuseppe Fusco. «A calibration method for odometry of mobile robots based on the least-squares technique: theory and experimental validation». En: *Robotics, IEEE Transactions on* 21.5 (2005), págs. 994-1004 (vid. pág. 284).

[10] Estefanía Argente, Vicent Botti y Vicente Julian. «GORMAS: An Organizational-Oriented Methodological Guideline for Open MAS». English. En: *Agent-Oriented Software Engineering X*. Ed. por Marie-Pierre Gleizes y JorgeJ. Gomez-Sanz. Vol. 6038. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, págs. 32-47. ISBN: 978-3-642-19207-4. DOI: 10.1007/978-3-642-19208-1_3 (vid. pág. 10).

[11] Leopoldo Armesto y col. «Probabilistic self-localization and mapping-an asynchronous multirate approach». En: *IEEE robotics & automation magazine* 15.2 (2008), págs. 77-88 (vid. pág. 127).

[12] Koestler Arthur. *The case of the Midwife Toad*. Arkana Books. 1971 (vid. pág. 15).

[13] Hajime Asama y col. «Collision avoidance among multiple mobile robots based on rules and communication». En: *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*. IEEE. 1991, págs. 1215-1220 (vid. pág. 234).

[14] Hazem Ali Attia. «Dynamic model of multi-rigid-body systems based on particle dynamics with recursive approach». En: *Journal of Applied Mathematics* 2005.4 (2005), págs. 365-382 (vid. págs. 291, 296, 304).

[15] Alexander Bahr, John J Leonard y Maurice F Fallon. «Cooperative localization for autonomous underwater vehicles». En: *The International Journal of Robotics Research* 28.6 (2009), págs. 714-728 (vid. pág. 154).

[16] Yaakov Bar-Shalom. «Update with out-of-sequence measurements in tracking: exact solution». En: *IEEE Transactions on Aerospace and Electronic Systems* 38.3 (2002), págs. 769-777 (vid. pág. 127).

[17] Fabio Bellifemine, Agostino Poggi y Giovanni Rimassa. «JADE-A FIPA-compliant agent framework». En: *Proceedings of PAAM*. Vol. 99. 97-108. London. 1999, pág. 33 (vid. págs. 3, 10).

[18] Fabio Bellifemine y col. «Jade administrator's guide». En: *TILab (February 2006)* (2003) (vid. pág. 22).

[19] Carole Bernon y col. «Adelfe: A methodology for adaptive multi-agent systems engineering». En: *Engineering Societies in the Agents World III*. Springer, 2003, págs. 156-169 (vid. pág. 10).

[20] Gildas Besançon. «An overview on observer tools for nonlinear systems». En: *Nonlinear observers and applications*. Springer, 2007, págs. 1-33 (vid. pág. 58).

[21] Paul J Besl y Neil D McKay. «Method for registration of 3-D shapes». En: *Robotics-DL tentative*. International Society for Optics y Photonics. 1992, págs. 586-606 (vid. pág. 102).

[22] Margrit Betke y Leonid Gurvits. «Mobile robot localization using landmarks». En: *IEEE transactions on robotics and automation* 13.2 (1997), págs. 251-263 (vid. págs. 160, 161).

[23] M Bliesener y col. «Festo Robotino Manual». En: *Denkendorf: Festo Didactic GmbH & Co. KG* (2007) (vid. pág. 117).

[24] Theodor Borangiu y col. «Multitasking robot-vision for supply holon execution in intelligent manufacturing». En: *Robotics in Alpe-Adria-Danube Region (RAAD), 2014 23rd International Conference on*. 2014, págs. 1-8. DOI: 10.1109/RAAD.2014.7002227 (vid. pág. 17).

[25] Johann Borenstein y Yoram Koren. «The vector field histogram-fast obstacle avoidance for mobile robots». En: *IEEE Transactions on Robotics and Automation* 7.3 (1991), págs. 278-288 (vid. pág. 233).

[26] Jeffrey M Bradshaw y col. «Making agents acceptable to people». En: *Intelligent Technologies for Information Analysis*. Springer, 2004, págs. 361-406 (vid. pág. 10).

[27] Valentino Braitenberg. *Vehicles: Experiments in synthetic psychology*. MIT press, 1986 (vid. págs. 217, 233).

[28] R. W. Brennan y D. H. Norrie. «Agents, Holons and Functions Blocks: Distributed Intelligent Control in Manufacturing». En: *Journal of Applied Systems Studies Special Issue on Industrial Applications of Multi-Agent and Holonic Systems*. 2 (2001), págs. 1-19 (vid. pág. 16).

[29] Damiano Brigo, Bernard Hanzon, François Le Gland y col. «Approximate nonlinear filtering by projection on exponential manifolds of densities». En: *Bernoulli* 5.3 (1999), págs. 495-534 (vid. pág. 59).

[30] Robert Grover Brown y Patrick Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering with Matlab Exercises and Solutions*. 3rd Edition. John Wiley & Sons, 1997 (vid. pág. 70).

[31] Davide Brugali y Patrizia Scandurra. «Component-based robotic engineering (part i)[tutorial]». En: *Robotics & Automation Magazine, IEEE* 16.4 (2009), págs. 84-96 (vid. págs. 2, 13).

[32] Herman Bruyninckx. «Open robot control software: the OROCOS project». En: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. Vol. 3. IEEE. 2001, págs. 2523-2528 (vid. pág. 14).

[33] Stefan Bussmann y Duncan C McFarlane. «Rationales for holonic manufacturing control». En: *Proc. of Second Int. Workshop on Intelligent Manufacturing Systems*. 1999, págs. 177-184 (vid. pág. 16).

[34] Giovanni Caire. «JADE tutorial: JADE programming for beginners». En: *Documentación de JADE* 3 (2003) (vid. págs. 23, 24, 30).

[35] Giuseppe C Calafiore, Luca Carlone y Mingzhu Wei. «A distributed technique for localization of agent formations from relative range measurements». En: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 42.5 (2012), págs. 1065-1076 (vid. pág. 157).

[36] Daniele Calisi y col. «OpenRDK: a modular framework for robotic software development». En: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE. 2008, págs. 1872-1877 (vid. pág. 14).

[37] David Camacho, Fernando Fernandez y Miguel A Rodelgo. «Roboskeleton: An architecture for coordinating robot soccer agents». En: *Engineering Applications of Artificial Intelligence* 19.2 (2006), págs. 179-188 (vid. pág. 18).

[38] Ricardo Carona, A Pedro Aguiar y Jose Gaspar. «Control of unicycle type robots tracking, path following and point stabilization». En: (2008) (vid. pág. 289).

[39] Haoyao Chen y col. «Localization for multirobot formations in indoor environment». En: *IEEE/ASME Transactions on Mechatronics* 15.4 (2010), págs. 561-574 (vid. pág. 157).

[40] Min Chen y col. «Mobile agent based wireless sensor networks». En: *Journal of computers* 1.1 (2006), págs. 14-21 (vid. pág. 219).

[41] Yang Chen y Gérard Medioni. «Object modeling by registration of multiple range images». En: *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*. IEEE. 1991, págs. 2724-2729 (vid. pág. 102).

[42] Andrea Cherubini y François Chaumette. «Visual navigation of a mobile robot with laser-based collision avoidance». En: *The International Journal of Robotics Research* 32.2 (2013), págs. 189-205 (vid. pág. 231).

[43] Jinwoo Choi y col. «Autonomous topological modeling of a home environment and topological localization using a sonar grid map». En: *Autonomous Robots* 30.4 (2011), págs. 351-368 (vid. pág. 55).

[44] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005 (vid. pág. 157).

[45] J. Christensen. *Agent-Based Manufacturing. Advances in the Holonic Approach*. Springer Verlag. Architecture and Its Implementation. 2003 (vid. pág. 15).

[46] Marius Staicu Ciprian Candea y Boldur Barbat. «Holon - Like Approach for Robotic Soccer». En: . 1 (2001), págs. 1-8 (vid. pág. 18).

[47] David T Cole y col. «System development and demonstration of a cooperative UAV team for mapping and tracking». En: *The International Journal of Robotics Research* 29.11 (2010), págs. 1371-1399 (vid. pág. 154).

[48] Peter Corke, Ron Peterson y Daniela Rus. «Localization and navigation assisted by networked cooperating sensors and robots». En: *The International Journal of Robotics Research* 24.9 (2005), págs. 771-786 (vid. pág. 154).

[49] A Cuenca. «Modelado, análisis y diseño de sistemas de control con muestreo no convencional». Tesis doct. Thesis. Department of Systems Engineering y Control. Politechnical University of Valencia.(In Spanish), 2004 (vid. pág. 127).

[50] Colin Das. *Calibration of Kinematic Odometric Parameters for Differential Drive Mobile Robots: An Overview*. 2010 (vid. pág. 284).

[51] Celso De La Cruz y Ricardo Carelli. «Dynamic model based formation control and obstacle avoidance of multi-robot systems». En: *Robotica* 26.03 (2008), págs. 345-356 (vid. págs. 289, 304).

[52] Pierre Del Moral. *Feynman-Kac Formulae*. Springer, 2004 (vid. pág. 60).

[53] Pierre Del Moral. «Non-linear filtering: interacting particle resolution». En: *Markov processes and related fields* 2.4 (1996), págs. 555-581 (vid. pág. 60).

[54] Frank Dellaert y col. «Monte carlo localization for mobile robots». En: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 2. IEEE. 1999, págs. 1322-1328 (vid. págs. 101, 202).

[55] Yoann Dieudonné, Ouiddad Labbani-Igbida y Franck Petit. «Deterministic robot-network localization is hard». En: *IEEE Transactions on Robotics* 26.2 (2010), págs. 331-339 (vid. págs. 160, 161).

[56] Matthew Dunbabin y col. «Experiments with cooperative control of underwater robots». En: *The International Journal of Robotics Research* 28.6 (2009), págs. 815-833 (vid. pág. 154).

[57] Gia Dvali, Gregory Gabadadze y Massimo Porrati. «4D gravity on a brane in 5D Minkowski space». En: *Physics Letters B* 485.1 (2000), págs. 208-214 (vid. pág. 237).

[58] Sean P Engelson y Drew V McDermott. «Error correction in mobile robot map learning». En: *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*. IEEE. 1992, págs. 2555-2560 (vid. pág. 101).

[59] João Sena Esteves, Adriano Carvalho y Carlos Couto. «Generalized geometric triangulation algorithm for mobile robot absolute self-localization». En: *Industrial Electronics, 2003. ISIE'03. 2003 IEEE International Symposium on*. Vol. 1. IEEE. 2003, págs. 346-351 (vid. págs. 127, 161).

[60] Alessandro Farinelli, Giorgio Grisetti y Luca Iocchi. «Design and implementation of modular software for programming mobile robots». En: *International Journal of Advanced Robotic Systems* 3.1 (2006), págs. 37-42 (vid. págs. 2, 13).

[61] Rafael Fierro y col. «A framework and architecture for multi-robot coordination». En: *The International Journal of Robotics Research* 21.10-11 (2002), págs. 977-995 (vid. pág. 157).

[62] Tim Finin y col. «KQML as an agent communication language». En: *Proceedings of the third international conference on Information and knowledge management*. ACM. 1994, págs. 456-463 (vid. pág. 10).

[63] Klaus Fischer, Michael Schillo y Jorg Siekmann. «Holonic multiagent systems: A foundation for the organisation of multiagent systems». En: *Holonic*

and Multi-Agent Systems for Manufacturing. Springer, 2003, págs. 71-80 (vid. pág. 16).

[64] JF Flores-Resendiz y col. «Finite-Time Formation Control without Collisions for Multiagent Systems with Communication Graphs Composed of Cyclic Paths». En: *Mathematical Problems in Engineering* 2015 (2015) (vid. pág. 234).

[65] Dieter Fox y col. «Monte carlo localization: Efficient position estimation for mobile robots». En: *AAAI/IAAI* 1999 (1999), págs. 343-349 (vid. pág. 202).

[66] Antonio Franchi, Giuseppe Oriolo y Paolo Stegagno. «Mutual localization in a multi-robot system with anonymous relative position measures». En: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, págs. 3974-3980 (vid. pág. 160).

[67] Stan Franklin y Art Graesser. «Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents». En: *Intelligent agents III agent theories, architectures, and languages*. Springer, 1997, págs. 21-35 (vid. pág. 11).

[68] Marc Freese y col. «Virtual Robot Experimentation Platform V-REP: A Versatile 3D Robot Simulator». English. En: *Simulation, Modeling, and Programming for Autonomous Robots*. Ed. por Noriaki Ando y col. Vol. 6472. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, págs. 51-62. ISBN: 978-3-642-17318-9. DOI: 10.1007/978-3-642-17319-6_8 (vid. pág. 42).

[69] Ran Gal y Daniel Cohen-Or. «Salient geometric features for partial shape matching and similarity». En: *ACM Transactions on Graphics (TOG)* 25.1 (2006), págs. 130-150 (vid. pág. 102).

[70] Shuzhi Sam Ge. *Autonomous mobile robots: sensing, control, decision making and applications*. Vol. 22. CRC press, 2006 (vid. pág. 154).

[71] Natasha Gelfand y col. «Robust global registration». En: *Symposium on geometry processing*. Vol. 2. 3. 2005, pág. 5 (vid. pág. 102).

[72] Mohinder S. Grewal y Angus P. Andrews. *Kalman Filtering, Theory and Practice Using Matlab*. John Wiley & Sons, 2001 (vid. págs. 55, 70).

[73] Ramiro Velázquez Guerrero. «Dinámica y Control de Sillas de Ruedas Robóticas, Avances en Ingeniería Electrónica». En: *por M. Magos & R.*

Campos M. Alcaraz. UAM-UdG (México, DF) (2010), págs. 151-164 (vid. pág. 289).

[74] María Guinaldo y col. «A mobile robots experimental environment with event-based wireless communication». En: *Sensors* 13.7 (2013), págs. 9396-9413 (vid. pág. 129).

[75] Ritu Gupta y Gaurav Kansal. «A Survey on Comparative Study of Mobile Agent Platforms.» En: *International Journal of Engineering Science & Technology* 3.3 (2011) (vid. pág. 21).

[76] *HMS Press Release, Revision 3.6, HMS Server* (vid. pág. 15).

[77] Bohyung Han y col. «Incremental density approximation and kernel-based bayesian filtering for object tracking». En: *CVPR (1)*. 2004, págs. 638-644 (vid. pág. 59).

[78] Seung-Beom Han, Jong-Hwan Kim y Hyun Myung. «Landmark-based particle localization algorithm for mobile robots with a fish-eye vision system». En: *IEEE/ASME Transactions on Mechatronics* 18.6 (2013), págs. 1745-1756 (vid. pág. 161).

[79] Vincent Hilaire, Abder Koukam y Sebastian Rodriguez. «An adaptative agent architecture for holonic multi-agent systems». En: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 3.1 (2008), pág. 2 (vid. pág. 16).

[80] Robert W Hogg y col. «Algorithms and sensors for small robot path following». En: *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 4. IEEE. 2002, págs. 3850-3857 (vid. pág. 157).

[81] Qi-Xing Huang y col. «Reassembling fractured objects by geometric matching». En: *ACM Transactions on Graphics (TOG)* 25.3 (2006), págs. 569-578 (vid. pág. 102).

[82] Lars Imsland y col. *Nonlinear observer for vehicle velocity with friction and road bank angle adaptation-validation and comparison with an extended Kalman filter*. Inf. téc. SAE Technical Paper, 2007 (vid. pág. 60).

[83] Pablo Iñigo-Blasco y col. «Robotics software frameworks for multi-agent robotic systems development». En: *Robotics and Autonomous Systems* 60.6 (2012), págs. 803-821 (vid. págs. 14, 21).

[84] D. McFarlane J. Jarvis R. Ronnquist y L. Jain. «A team-based holonic approach to robotic assembly cell control». En: *Journal of Network and Computer Applications* 29 (2006). Innovations in agent collaboration, págs. 160 -176. ISSN: 1084-8045. DOI: http://dx.doi.org/10.1016/j.jnca.2004.10.001 (vid. pág. 17).

[85] Pablo Jiménez, Federico Thomas y Carme Torras. «3D collision detection: a survey». En: *Computers & Graphics* 25.2 (2001), págs. 269-285 (vid. pág. 232).

[86] Rudolph Emil Kalman. «A new approach to linear filtering and prediction problems». En: *Journal of Fluids Engineering* 82.1 (1960), págs. 35-45 (vid. pág. 58).

[87] Jon Klein. «Breve: a 3d environment for the simulation of decentralized systems and artificial life». En: *Proceedings of the eighth international conference on Artificial life*. 2003, págs. 329-334 (vid. pág. 42).

[88] James T Klosowski y col. «Efficient collision detection using bounding volume hierarchies of k-DOPs». En: *IEEE transactions on Visualization and Computer Graphics* 4.1 (1998), págs. 21-36 (vid. págs. 232, 233).

[89] Franziska Klügl y Frank Puppe. «The multi-agent simulation environment sesam». En: *Proceedings des Workshops Simulation in Knowledge-based Systems, volume tr-ri-98-194 of Reihe Informatik, Paderborn*. Citeseer. 1998 (vid. pág. 42).

[90] Arthur Koestler. *The Ghost in the Machine*. Arkana Books, London, UK., 1967 (vid. pág. 15).

[91] Kurt Konolige. «A gradient method for realtime robot control». En: *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*. Vol. 1. IEEE. 2000, págs. 639-646 (vid. pág. 207).

[92] David Kortenkamp y Reid Simmons. «Robotic systems architectures and programming». En: *Springer Handbook of Robotics*. Springer, 2008, págs. 187-206 (vid. págs. 2, 13).

[93] Krzysztof Kozłowski. «Robot motion and control(recent developments)». En: *Lecture notes in control and information sciences* (2006) (vid. pág. 157).

[94] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006 (vid. pág. 157).

[95] Emmett Lalish y Kristi A Morgansen. «Decentralized reactive collision avoidance for multivehicle systems». En: *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*. IEEE. 2008, págs. 1218-1224 (vid. pág. 234).

[96] Emmett Lalish y Kristi A Morgansen. «Distributed reactive collision avoidance». En: *Autonomous Robots* 32.3 (2012), págs. 207-226 (vid. pág. 234).

[97] Gilad Langer. *HoMuCS-A methodology and architecture for Holonic Multicell Control Systems*. Ed. por Technical University of Denmark Production Technology Department of Manufacturing Engineering. IPT, DTU, 1999. DOI: 8790855000 (vid. pág. 16).

[98] Eric Larsen y col. *Fast proximity queries with swept sphere volumes*. Inf. téc. Technical Report TR99-018, Department of Computer Science, University of North Carolina, 1999 (vid. pág. 232).

[99] Thomas Larsson y Tomas Akenine-Möller. «A dynamic bounding volume hierarchy for generalized collision detection». En: *Computers & Graphics* 30.3 (2006), págs. 450-459 (vid. pág. 233).

[100] Olivier Lefebvre y col. «Obstacles avoidance for car-like robots integration and experimentation on two robots». En: *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. Vol. 5. IEEE. 2004, págs. 4277-4282 (vid. pág. 157).

[101] Rafał Leszczyna. «Evaluation of agent platforms». En: *European Commission, Joint Research Centre, Institute for the Protection and Security of the Citizen, Ispra, Italy, Tech. Rep* (2004) (vid. pág. 21).

[102] Keith YK Leung, Timothy D Barfoot y Hugh HT Liu. «Decentralized localization of sparsely-communicating robot networks: A centralized-equivalent approach». En: *IEEE Transactions on Robotics* 26.1 (2010), págs. 62-77 (vid. pág. 154).

[103] Xinju Li e Igor Guskov. «Multiscale Features for Approximate Alignment of Point-based Surfaces.» En: *Symposium on geometry processing*. Vol. 2. Citeseer. 2005 (vid. pág. 102).

[104] Vincent Louis y Thierry Martinez. «An operational model for the FIPA-ACL semantics». En: *Agent Communication II*. Springer, 2006, págs. 1-14 (vid. págs. 3, 22).

[105] Tomás Lozano-Pérez y Michael A Wesley. «An algorithm for planning collision-free paths among polyhedral obstacles». En: *Communications of the ACM* 22.10 (1979), págs. 560-570 (vid. pág. 232).

[106] Feng Lu y Evangelos Milios. «Globally consistent range scan alignment for environment mapping». En: *Autonomous robots* 4.4 (1997), págs. 333-349 (vid. págs. 200, 203).

[107] Sean Luke y col. «Mason: A new multi-agent simulation toolkit». En: *Proceedings of the 2004 swarmfest workshop*. Vol. 8. 2004 (vid. pág. 42).

[108] Vladimir J. Lumelsky y KR Harinarayan. «Decentralized motion planning for multiple mobile robots: The cocktail party model». En: *Robot colonies*. Springer, 1997, págs. 121-135 (vid. pág. 233).

[109] Vladimir J Lumelsky y Alexander A Stepanov. «Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape». En: *Algorithmica* 2.1-4 (1987), págs. 403-430 (vid. pág. 157).

[110] Martin Lundgren. «Path tracking and obstacle avoidance for a miniature robot». En: *Umeå University, Umeå, Master Thesis* (2003) (vid. pág. 157).

[111] Kristijan Macek, Ivan Petrovic y Roland Siegwart. «A control method for stable and smooth path following of mobile robots». En: *None*. LSA-CONF-2005-035. 2005 (vid. pág. 157).

[112] Raj Madhavan y Hugh F Durrant-Whyte. «Natural landmark-based autonomous vehicle navigation». En: *Robotics and Autonomous Systems* 46.2 (2004), págs. 79-95 (vid. pág. 160).

[113] Guoqiang Mao, Barış Fidan y Brian DO Anderson. «Wireless sensor network localization techniques». En: *Computer networks* 51.10 (2007), págs. 2529-2553 (vid. pág. 160).

[114] Vladimír Maríık, JL Martinez Lastra y Petr Skobelev. «Industrial Applications of Holonic and Multi-Agent Systems». En: *6th International Conference, HoloMAS*. Springer. 2013 (vid. pág. 16).

[115] Leonardo Marin y col. «Event-based localization in ackermann steering limited resource mobile robots». En: *IEEE/ASME Transactions on Mechatronics* 19.4 (2014), págs. 1171-1182 (vid. pág. 62).

[116] Leonardo Marín y col. «Multi sensor fusion framework for indoor-outdoor localization of limited resource mobile robots». En: *Sensors* 13.10 (2013), págs. 14133-14160 (vid. pág. 62).

[117] Agostino Martinelli, Frederic Pont y Roland Siegwart. «Multi-robot localization using relative observations». En: *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE. 2005, págs. 2797-2802 (vid. pág. 154).

[118] Ismet Erkmen Mehmet Durna Aydan Erkmen. «Self-Localization of a Holon in the Reconfiguration Task Space of a Robotic Colony». En: *International Conference on Robotics and Automation* 1 (2000), págs. 1-5 (vid. pág. 18).

[119] Mehran Mesbahi y Magnus Egerstedt. *Graph theoretic methods in multiagent networks*. Princeton University Press, 2010 (vid. pág. 19).

[120] Giorgio Metta, Paul Fitzpatrick y Lorenzo Natale. «YARP: yet another robot platform». En: *International Journal on Advanced Robotics Systems* 3.1 (2006), págs. 43-48 (vid. pág. 14).

[121] Olivier Michel, Fabien Rohrer y Nicolas Heiniger. «Cyberbotics' Robot Curriculum». En: (2014) (vid. págs. 90, 288).

[122] Mihail Mihaylov, Karl Tuyls y Ann Nowé. «A decentralized approach for convention emergence in multi-agent systems». English. En: *Autonomous Agents and Multi-Agent Systems* 28.5 (2014), págs. 749-778. ISSN: 1387-2532. DOI: 10.1007/s10458-013-9240-2 (vid. pág. 21).

[123] Michael Montemerlo, Nicholas Roy y Sebastian Thrun. «Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit». En: *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. Vol. 3. IEEE. 2003, págs. 2436-2441 (vid. pág. 200).

[124] Yamamoto Mori. «Maeda». En: *Scanning Laser Range Finder URG-04LX-UG01 Specifications. Np: Hokuyo Automic Co, LTD* 20 (2009) (vid. pág. 118).

[125] Anastasios I Mourikis y Stergios I Roumeliotis. «Optimal sensor scheduling for resource-constrained localization of mobile robot formations». En: *IEEE Transactions on Robotics* 22.5 (2006), págs. 917-931 (vid. pág. 154).

[126] Anastasios I Mourikis y Stergios I Roumeliotis. «Predicting the performance of cooperative simultaneous localization and mapping (C-SLAM)». En: *The*

International Journal of Robotics Research 25.12 (2006), págs. 1273-1286 (vid. pág. 157).

[127] Kevin P Murphy y col. «Bayesian Map Learning in Dynamic Environments.» En: *NIPS*. 1999, págs. 1015-1021 (vid. pág. 197).

[128] Robien R. Murphy. *Introduction to AI Robotics*. 2000 (vid. pág. 11).

[129] Rudy Negenborn. «Robot localization and kalman filters». Tesis doct. Citeseer, 2003 (vid. pág. 101).

[130] S Noga. «Kinematics and dynamics of some selected two-wheeled mobile robots». En: *Archives of Civil and Mechanical Engineering* 6.3 (2006), págs. 55-70 (vid. págs. 289, 304).

[131] Patrick D O Brien y Richard C Nicol. «FIPA towards a standard for software agents». En: *BT Technology Journal* 16.3 (1998), págs. 51-59 (vid. pág. 10).

[132] James Odell. «OMG Agent Standardization». English. En: *Agent-Based Technologies and Applications for Enterprise Interoperability*. Vol. 98. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2012, págs. 1-9. ISBN: 978-3-642-28562-2. DOI: 10.1007/978-3-642-28563-9_1 (vid. pág. 10).

[133] Anders Orebäck y Henrik I Christensen. «Evaluation of architectures for mobile robotics». En: *Autonomous robots* 14.1 (2003), págs. 33-49 (vid. pág. 13).

[134] Djamila Ouelhadj. «A multi-agent system for the integrated dynamic scheduling of steel production». Tesis doct. Citeseer, 2003 (vid. pág. 21).

[135] Juan Pavón y Jorge Gómez-Sanz. «Agent oriented software engineering with INGENIAS». En: *Multi-Agent Systems and Applications III*. Springer, 2003, págs. 394-403 (vid. pág. 10).

[136] Douglas L Peckover. *Intelligent agents for electronic commerce*. US Patent 6,119,101. 2000 (vid. pág. 11).

[137] Vincent Pierlot y Marc Van Droogenbroeck. «A new three object triangulation algorithm for mobile robot positioning». En: *IEEE Transactions on Robotics* 30.3 (2014), págs. 566-577 (vid. pág. 161).

[138] R Piza y col. «Kalman filtering applied to Profibus-DP systems. Multirate control systems with delayed signals». En: *Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE*. IEEE. 2008, págs. 2905-2910 (vid. pág. 127).

[139] Amanda Prorok, Alexander Bahr y Alcherio Martinoli. «Low-cost collaborative localization for large-scale multi-robot systems». En: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. Ieee. 2012, págs. 4236-4241 (vid. págs. 154, 160).

[140] Jim Pugh y col. «A fast onboard relative positioning module for multirobot systems». En: *IEEE/ASME Transactions on Mechatronics* 14.2 (2009), págs. 151-162 (vid. pág. 160).

[141] Morgan Quigley y col. «ROS: an open-source Robot Operating System». En: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, pág. 5 (vid. págs. 3, 14, 196).

[142] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011 (vid. págs. 301, 302, 308).

[143] Stéphane Redon, Abderrahmane Kheddar y Sabine Coquillart. «Fast continuous collision detection between rigid bodies». En: *Computer graphics forum*. Vol. 21. 3. Wiley Online Library. 2002, págs. 279-287 (vid. pág. 233).

[144] Stephane Redon y col. «Fast continuous collision detection for articulated models». En: *Journal of Computing and Information Science in Engineering* 5.2 (2005), págs. 126-137 (vid. pág. 233).

[145] Ioannis Rekleitis, Gregory Dudek y Evangelos Milios. «Multi-robot collaboration for robust exploration». En: *Annals of Mathematics and Artificial Intelligence* 31.1-4 (2001), págs. 7-40 (vid. pág. 154).

[146] Ioannis Rekleitis y col. «Efficient boustrophedon multi-robot coverage: an algorithmic approach». En: *Annals of Mathematics and Artificial Intelligence* 52.2-4 (2008), págs. 109-142 (vid. pág. 154).

[147] Paul Richmond y col. «High performance cellular level agent-based simulation with FLAME for the GPU». En: *Briefings in bioinformatics* 11.3 (2010), págs. 334-347 (vid. pág. 42).

[148] Sebastian Rodriguez, Vincent Hilaire y Abder Koukam. «Formal specification of holonic multi-agent systems framework». En: *Computational Science ICCS 2005*. Springer, 2005, págs. 719-726 (vid. pág. 16).

[149] Jonathan A Rogge y Dirk Aeyels. «Multi-robot coverage to locate fixed and moving targets». En: *2009 IEEE Control Applications,(CCA) & Intelligent Control,(ISIC)*. IEEE. 2009, págs. 902-907 (vid. pág. 154).

[150] Stergios I Roumeliotis y George A Bekey. «Distributed multirobot localization». En: *IEEE Transactions on Robotics and Automation* 18.5 (2002), págs. 781-795 (vid. págs. 154, 156, 176, 184).

[151] Stergios I Roumeliotis y Ioannis M Rekleitis. «Propagation of uncertainty in cooperative multirobot localization: Analysis and experimental results». En: *Autonomous Robots* 17.1 (2004), págs. 41-54 (vid. pág. 154).

[152] Szymon Rusinkiewicz y Marc Levoy. «Efficient variants of the ICP algorithm». En: *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*. IEEE. 2001, págs. 145-152 (vid. pág. 102).

[153] Stuart Russell y Peter Norvig. «AI a modern approach». En: *Learning* 2.3 (2005), pág. 4 (vid. págs. 10-12).

[154] Tuomas Sandholm. «An implementation of the contract net protocol based on marginal cost calculations». En: *AAAI*. Vol. 93. 1993, págs. 256-262 (vid. pág. 11).

[155] Radim Sara, Ikuko Shimizu Okatani y Akihiro Sugimoto. «Globally convergent range image registration by graph kernel algorithm». En: *3-D Digital Imaging and Modeling, 2005. 3DIM 2005. Fifth International Conference on*. IEEE. 2005, págs. 377-384 (vid. pág. 102).

[156] Nilanjan Sarkar, Xiaoping Yun y Vijay Kumar. «Control of mechanical systems with rolling constraints application to dynamic control of mobile robots». En: *The International Journal of Robotics Research* 13.1 (1994), págs. 55-69 (vid. pág. 289).

[157] Michael Schillo. «Self-organization and adjustable autonomy: two sides of the same coin?» En: *Connection Science* 14.4 (2002), págs. 345-359 (vid. pág. 17).

[158] Michael Schillo y Klaus Fischer. «Holonic multiagent systems». En: *Manufacturing Systems* 8.13 (2002), págs. 538-550 (vid. pág. 16).

[159] Georg S Seyboth, Dimos V Dimarogonas y Karl H Johansson. «Event-based broadcasting for multi-agent average consensus». En: *Automatica* 49.1 (2013), págs. 245-252 (vid. pág. 129).

[160] Ying Shan y col. «Linear model hashing and batch ransac for rapid and accurate object recognition». En: *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*. Vol. 2. IEEE. 2004, págs. II-121 (vid. pág. 102).

[161] Onn M Shehory y Arnon Sturm. *Agent-oriented software engineering: reflections on architectures, methodologies, languages, and frameworks*. Springer, 2014 (vid. pág. 10).

[162] Weiming Shen y col. «Applications of agent-based systems in intelligent manufacturing: An updated review». En: *Advanced engineering INFORMATICS* 20.4 (2006), págs. 415-431 (vid. pág. 16).

[163] Roland Siegwart, Illah Reza Nourbakhsh y Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011 (vid. págs. 55, 157, 288).

[164] RH Silsbee. «Focusing in collision problems in solids». En: *Journal of Applied Physics* 28.11 (1957), págs. 1246-1250 (vid. pág. 232).

[165] Thierry Siméon, J-P Laumond y Carole Nissoux. «Visibility-based probabilistic roadmaps for motion planning». En: *Advanced Robotics* 14.6 (2000), págs. 477-493 (vid. pág. 233).

[166] Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley y Sons, 2006 (vid. págs. 56, 58, 59, 70, 127, 130).

[167] I. Skog y P. Handel. «In-car positioning and navigation technologies - a survey». En: *Intelligent Transportation Systems, IEEE Transactions.* 10 (2009), págs. 4-21 (vid. págs. 5, 55).

[168] Brian Smith y col. «Multi-robot deployment and coordination with embedded graph grammars». En: *Autonomous Robots* 26.1 (2009), págs. 79-98 (vid. pág. 19).

[169] Randall Smith, Matthew Self y Peter Cheeseman. «Estimating uncertain spatial relationships in robotics». En: *Autonomous robot vehicles*. Springer, 1990, págs. 167-193 (vid. pág. 101).

[170] Joan Sola. «Simulataneous localization and mapping with the extended Kalman filter». En: *unpublished. Available: http://www. joansola. eu/JoanSola/eng/JoanSola. html* (2013) (vid. pág. 131).

[171] Joan Sola. «Towards visual localization, mapping and moving objects tracking by a mobile robot: a geometric and probabilistic approach». Tesis doct. Institut National Polytechnique de Toulouse-INPT, 2007 (vid. pág. 131).

[172] Bussmann Stefan. «An Agent-Oriented Architecture for Holonic Manufacturing Control». En: *Proceedings of First International Workshop on IMS, Lausanne, Switzerland* 1 (1998), págs. 1-12 (vid. pág. 16).

[173] Ashley Stroupe y col. «Sustainable cooperative robotic technologies for human and robotic outpost infrastructure construction and maintenance». En: *Autonomous Robots* 20.2 (2006), págs. 113-123 (vid. pág. 55).

[174] Gavin R. Sun Zhaohao Finnie. *Intelligent Techniques in E-Commerce*. Springer-Verlag Berlin Heidelberg, 2004 (vid. pág. 21).

[175] Ashleigh Swingler y Silvia Ferrari. «On the duality of robot and sensor path planning». En: *52nd IEEE Conference on Decision and Control*. IEEE. 2013, págs. 984-989 (vid. pág. 231).

[176] Shigeo Takahashi, Yuriko Takeshima e Issei Fujishiro. «Topological volume skeletonization and its application to transfer function design». En: *Graphical Models* 66.1 (2004), págs. 24-49 (vid. pág. 232).

[177] CheeKuang Tam, Richard Bucknall y Alistair Greig. «Review of collision avoidance and path planning methods for ships in close range encounters». En: *The Journal of Navigation* 62.3 (2009), pág. 455 (vid. pág. 233).

[178] Andrei Rosu Theodor Borangiu Silviu Raileanu y Mihai Parlea. *Holonic Robot Control for Job Shop Assembly by Dynamic Simulation*. Ed. por Luiz Affonso Guedes. Programmable Logic Controller, 2010 (vid. pág. 17).

[179] D Fox W Burgard S Thrun, D Fox y W Burgard. «The dynamic window approach to collision avoidance». En: *IEEE Transactions on Robotics and Automation* 4 (1997), pág. 1 (vid. pág. 208).

[180] Sebastian Thrun y col. «Robust Monte Carlo localization for mobile robots». En: *Artificial intelligence* 128.1 (2001), págs. 99-141 (vid. pág. 202).

[181] Nikolas Trawny, Stergios I Roumeliotis y Georgios B Giannakis. «Cooperative multi-robot localization under communication constraints». En: *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE. 2009, págs. 4394-4400 (vid. pág. 154).

[182] Julian J Valencia-Jimenez y Antonio Fernandez-Caballero. «Holonic multiagent systems to integrate independent multi-sensor platforms in complex surveillance». En: *Video and Signal Based Surveillance, 2006. AVSS06. IEEE International Conference on*. IEEE. 2006, págs. 49-49 (vid. pág. 16).

[183] A. Valera, A. Soriano y M. Vallés. «Plataformas de Bajo Coste para la Realización de Trabajos Prácticos de Mecatrónica y Robótica». En: *Revista Iberoamericana de Automática e Informática Industrial {RIAI}* 11.4 (2014), págs. 363 -376. ISSN: 1697-7912. DOI: `http://dx.doi.org/10.1016/j.riai.2014.09.002` (vid. pág. 45).

[184] Angel Valera y col. «Application and evaluation of Lego NXT tool for Mobile Robot Control». En: (vid. pág. 45).

[185] Jur Van Den Berg y col. «Reciprocal n-body collision avoidance». En: *Robotics research*. Springer, 2011, págs. 3-19 (vid. pág. 234).

[186] Martín Velasco Villa, Eduardo Aranda Bricaire y Rodolfo Orosco Guerrero. «Discrete-time modeling and path-tracking for a wheeled mobile robot». En: *Computación y Sistemas* 13.2 (2009) (vid. pág. 284).

[187] Michal Pechoucek Vladimir Marik Robert W. Brennan. *Holonic and Multi-Agent Systems for Manufacturing: Second International Conference on Industrial Applications of Holonic and Multi-Agent Systems*. Springer Science y Business Media, 2005 (vid. pág. 16).

[188] Chris C Ward y Karl Iagnemma. «A dynamic-model-based wheel slip detector for mobile robots on outdoor terrain». En: *IEEE Transactions on Robotics* 24.4 (2008), págs. 821-831 (vid. págs. 289, 304).

[189] Charles W Warren. «Fast path planning using modified A* method». En: *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. IEEE. 1993, págs. 662-667 (vid. pág. 233).

[190] Greg Welch y Gary Bishop. «An Introduction to the Kalman Filter. 2006». En: *University of North Carolina: Chapel Hill, North Carolina, US* () (vid. pág. 55).

[191] Greg Welch y Gary Bishop. «An introduction to the Kalman filter». En: (2006) (vid. pág. 58).

[192] Glenn D White, Rajankumar M Bhatt y Venkat N Krovi. «Dynamic redundancy resolution in a nonholonomic wheeled mobile manipulator». En: *Robotica* 25.2 (2007), págs. 147-156 (vid. págs. 154, 289).

[193] Michael Winikoff. «JACk intelligent agents: An industrial strength platform». En: *Multi-Agent Programming*. Springer, 2005, págs. 175-193 (vid. pág. 10).

[194] Jo Yung Wong. *Theory of ground vehicles*. John Wiley & Sons, 2001 (vid. págs. 301-304, 308).

[195] M. Wooldridge y N. R. Jennings. *Intelligent agents: Theory and practice.* 1995 (vid. pág. 11).

[196] Michael Wooldridge, Nicholas R Jennings y David Kinny. «The Gaia methodology for agent-oriented analysis and design». En: *Autonomous Agents and Multi-Agent Systems* 3.3 (2000), págs. 285-312 (vid. pág. 10).

[197] Xuedong Yan, Stephen Richards y Xiaogang Su. «Using hierarchical tree-based regression model to predict train–vehicle crashes at passive highway-rail grade crossings». En: *Accident Analysis & Prevention* 42.1 (2010), págs. 64-74 (vid. pág. 233).

[198] P. Zarchan y H. Musoff. *Fundamentals of Kalman Filtering: A Practical Approach*. EngineeringPro collection v. 208. American Institute of Aeronautics y Astronautics, 2005. ISBN: 9781600864582 (vid. pág. 70).

[199] Xinyu Zhang y col. «Continuous collision detection for articulated models using taylor models and temporal culling». En: *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM. 2007, pág. 15 (vid. pág. 233).

[200] Ilan Zohar, Amit Ailon y Raul Rabinovici. «Mobile robot characterized by dynamic and kinematic equations and actuator dynamics: Trajectory tracking and related application». En: *Robotics and autonomous systems* 59.6 (2011), págs. 343-353 (vid. págs. 289, 304).