UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Un sistema multiagente para mejorar la localización de un robot NAO

Trabajo Fin de Máster

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Autor: Peñaranda Cebrián, Cristian

Tutores: Julián Inglada, Vicente Javier
Palanca Cámara, Javier

2015-2016

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

GTI IA Grupo de Tecnología Informática Inteligencia Artificial

Resumen

En este trabajo se plantea una posible solución a un problema existente para la localización de un robot móvil, concretamente el robot NAO. La necesidad de la localización del robot surgió a la hora de realizar diferentes proyectos donde el robot se debía desplazar por un entorno dinámico. En estos proyectos, se pudo observar que el movimiento del robot no era regular, debido a la imprecisión de sus sensores, entre otras cosas. Típicamente, esto se ha resuelto en entornos estáticos mediante la adición de sensores que ayudan al robot, pero esto no es útil en entornos dinámicos donde el robot se desplaza por distintas salas o zonas. La novedad de este escenario dinámico es que cada zona está compuesta por dispositivos de sensorización propios, diferentes a los dispositivos de sensorización de otras zonas, que informan de la posición en la que se encuentra el robot. En este proyecto, se va a desarrollar un sistema multiagente sobre la plataforma *SPADE* para mejorar la localización de un robot NAO. Para ello, se hará uso de las funcionalidades de notificación de presencia y protocolos de subscripción de *SPADE* para montar una red de amistad entre los dispositivos, compuestos por sensores, y los robots móviles.

Palabras clave: Sistemas Multiagente, Robótica, *SPADE*, NAO.

Abstract

In this work, we are going to solve a possible solution to an existing problem to mobile robot location, specifically the NAO robot location. The need of the NAO robot location appears when we performed several projects where the robot moves through an dynamic environment. In these projects, we observed that the robot movement wasn't regularly. Typically this has been resolved in static environments by adding sensors that help the robot but this is not useful in dynamic environments where the robot moves through different rooms or areas. The novelty of this dynamic scenario is that each zone is composed of own devices that reporting the position where the robot is. In this project, we develop a multi-agent system in *SPADE* to improve the NAO robot location. So, we are going to use presence notification and subscriptions protocols of *SPADE* to make a friendship network between sensors and mobile robots.

Keywords: Multi-Agent System, Robotics, *SPADE*, NAO.

Tabla de contenidos

Tabla de figuras

1. Introducción

En los últimos años, se ha podido observar una gran evolución de los robots móviles. Debido a su gran cantidad de componentes, los robots son máquinas muy complejas. Por ese motivo, es bastante común que tengan dificultades para llevar a cabo los objetivos deseados, necesitando cierta ayuda para completarlos de manera adecuada. En este trabajo, se va a utilizar el robot humanoide NAO, fabricado por la empresa francesa *Aldebaran Robotics*, con el fin de ayudar a conocer su posición tras realizar una serie de movimientos, ya que es posible que no se consiga alcanzar la posición final deseada debido, entre otras cosas, a la imprecisión de sus sensores internos. Para ello, se va a emplear un sistema multiagente para lograr el objetivo deseado ya que aporta la distribución, flexibilidad y dinamismo necesario para la resolución del problema planteado. Además, el sistema creado podría utilizarse para ayudar al desplazamiento de cualquier robot móvil.

1.1. Motivación

En estudios anteriores, se ha utilizado el robot NAO con diferentes propósitos, en los que se ha observado que no alcanza la posición deseada tras un desplazamiento por muy corto que este sea. Sin embargo, el robot cree que ha alcanzado la posición correcta [1, 2]. La diferencia entre la posición buscada y la posición errónea puede parecer pequeña en desplazamientos simples, pero cuando estos son de mayor complejidad, aumenta considerablemente, haciendo que no logre completar el objetivo requerido. En la figura 1, se puede observar la evolución del desplazamiento del a través de un sencillo ejemplo. Existen dos caminos, el seguido por el robot, que se encuentra en la parte superior, y el que el robot desea realizar, que se encuentra en la parte inferior. En cada iteración, el robot realiza un desplazamiento simple, y el conjunto de estos desplazamientos simples genera un desplazamiento complejo. Como se puede observar, el error de movimiento cometido por el robot va aumentando en función del número de movimientos que realiza.

Los errores cometidos durante el desplazamiento se deben principalmente a dos causas:

- **Pavimento**: Dependiendo del pavimento en el que se mueva el robot, éste se desplazará de forma más o menos precisa. En algunos pavimentos el robot se desliza demasiado, mientras que otros ejercen una fuerza de rozamiento sobre los pies más elevada dificultando el desplazamiento del robot.
- **Componentes del robot**: Al estar el robot compuesto de múltiples componentes, los pequeños errores se acumulan y, como consecuencia, el resultado se desvía del esperado.

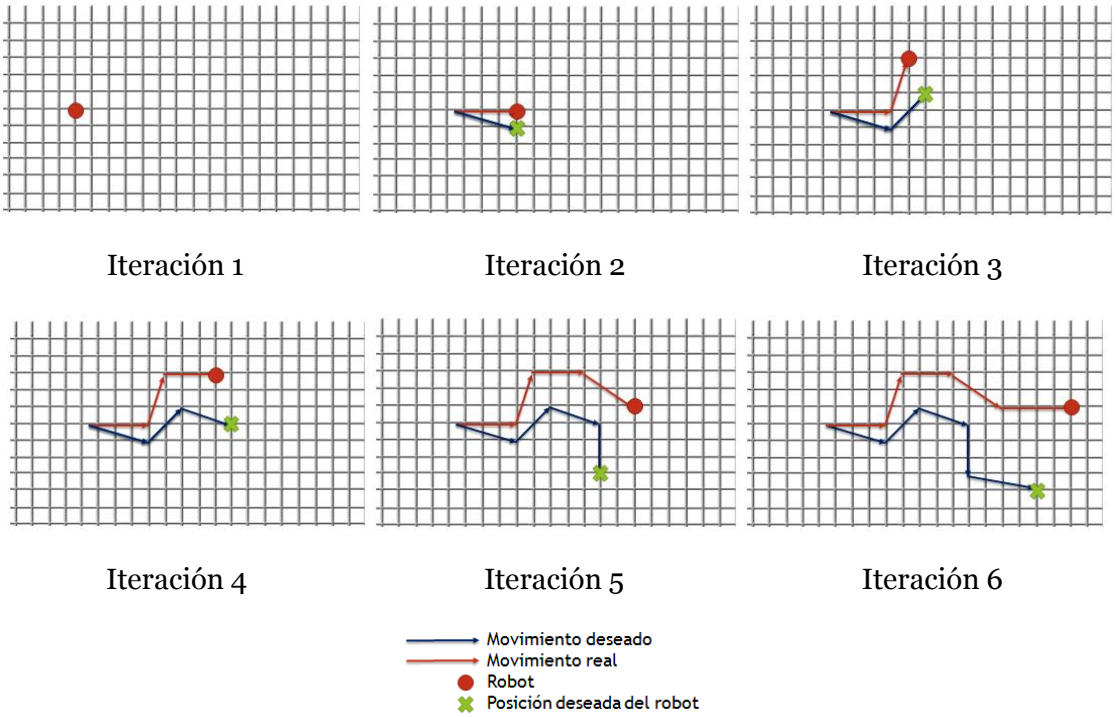


Figura 1 - Movimiento del robot NAO

Este problema ha sido abordado en diferentes estudios, donde se ha intentado localizar la posición del robot en un entorno estático de distintas formas: a través de cámaras incorporadas en el propio robot [3, 4], o por medio de cámaras fijas en el entorno [5, 6, 7].

El problema surge cuando el robot debe realizar tareas en distintas salas con condiciones cambiantes y no conocidas. Debido a esto, en el proyecto se plantea abordar este problema mediante una solución flexible que permita al robot conocer su posición apoyándose en dispositivos de sensorización externos, como cámaras, aún en el caso de ir cambiando su posición en distintas salas con distintos dispositivos de sensorización.

1.2. Objetivos

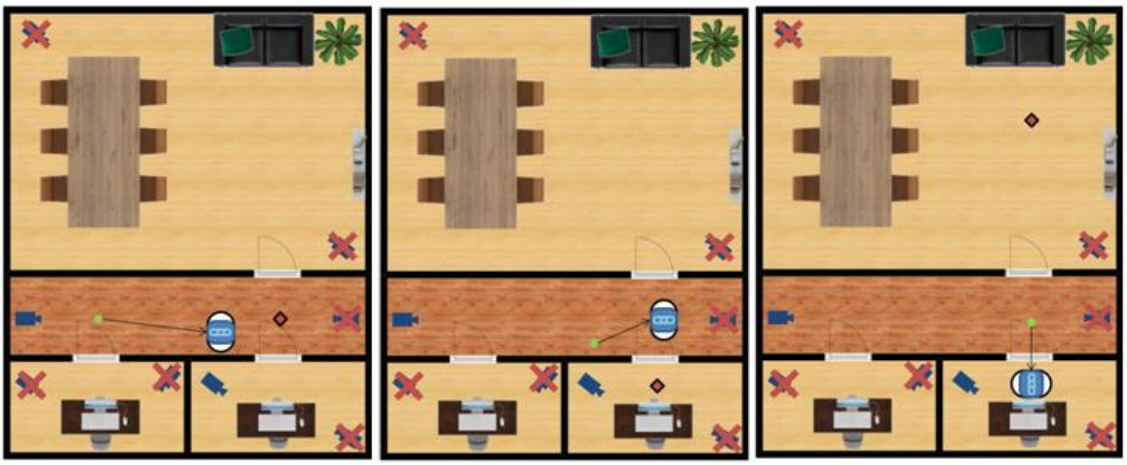
El objetivo principal de este trabajo es ayudar a alcanzar la posición deseada por el robot al desplazarse por diferentes zonas o salas. A diferencia de las soluciones existentes, que funcionan sobre un escenario estático [5, 6, 7], en este trabajo se plantea una solución flexible y distribuida mediante un sistema multiagente en el que se dispone de un agente que controla el robot NAO y de diferentes agentes que controlan a distintos dispositivos de sensorización que conocen la posición real o aproximada del robot. Además, el sistema multiagente permitirá que diferentes dispositivos de sensorización se incorporen o salgan de forma dinámica y transparente. Por otro lado,

la solución propuesta también permitirá que se pueda incorporar de forma dinámica cualquier tipo de dispositivo que puede dar información sobre la posición del robot.

En la figura 2 se puede observar un ejemplo donde el robot necesita desplazarse por diferentes habitaciones donde simulamos el comportamiento deseado. El robot formaría parte de un sistema multiagente para obtener ayuda en sus desplazamientos, al igual que otras cámaras que informan de su posición. En el ejemplo, existen dos tipos de cámaras, las que no están disponibles en el sistema multiagente, que aparecen con una cruz, y las que sí lo están, ya que, los dispositivos de sensorización pueden conectarse y desconectarse libremente al tratarse de un sistema flexible. En la iteración 1, el robot busca alcanzar la posición objetivo (rombo) con la ayuda del sistema multiagente, donde hay disponible una única cámara. En la siguiente iteración, el robot se ha desplazado a una posición incorrecta, por lo que no ha alcanzado su objetivo, pero gracias a la ayuda de la información de la cámara conoce que la posición en la que se encuentra no era la deseada y vuelve a realizar un movimiento para alcanzarlo. En la iteración 3, se puede observar que el robot ha logrado alcanzar su posición objetivo y que además aparecen tres nuevas cámaras que se incorporan al sistema multiagente. En la siguiente iteración, el robot consigue alcanzar la posición deseada en un sólo desplazamiento, pero aparece una nueva posición objetivo que no se puede lograr en un solo movimiento, por lo que lo realizará en dos movimientos. En el primer movimiento, correspondiente a la iteración 5, el robot abandona la sala en la que se encuentra y además las cámaras de esta sala dejan de participar en el sistema multiagente, ya que se trata de un sistema flexible se permite la incorporación y el abandono de los dispositivos de sensorización. En el siguiente movimiento, iteración 6, el robot no logra alcanzar la posición objetivo, pero gracias a la ayuda de las cámaras introducidas en el sistema multiagente logra alcanzarlo en la siguiente iteración. En la iteración 8, el robot consigue alcanzar la posición deseada en un único desplazamiento y aparece una nueva posición objetivo que podemos realizar en sólo un movimiento. Después de realizar el movimiento para alcanzar la posición deseada (iteración 9), se conecta una nueva cámara en el sistema multiagente y el robot detecta que la posición en la que se encuentra no es la deseada, por lo que intenta alcanzarla a través de un nuevo movimiento. Finalmente, en la iteración 10, el robot no ha logrado alcanzar aún su destino, debido a que la cámara que le ha informado no era precisa, pero se conecta una nueva cámara que complementa la información obtenida por la anterior cámara y ayudan a que el robot alcance el objetivo en el siguiente movimiento. Gracias a la ayuda planteada, el robot puede ir resituándose y realizar correctamente los desplazamientos deseados.

Iteración 1 Iteración 2 Leyenda

Iteración 3 Iteración 4 Iteración 5



Iteración 6 Iteración 7 Iteración 8

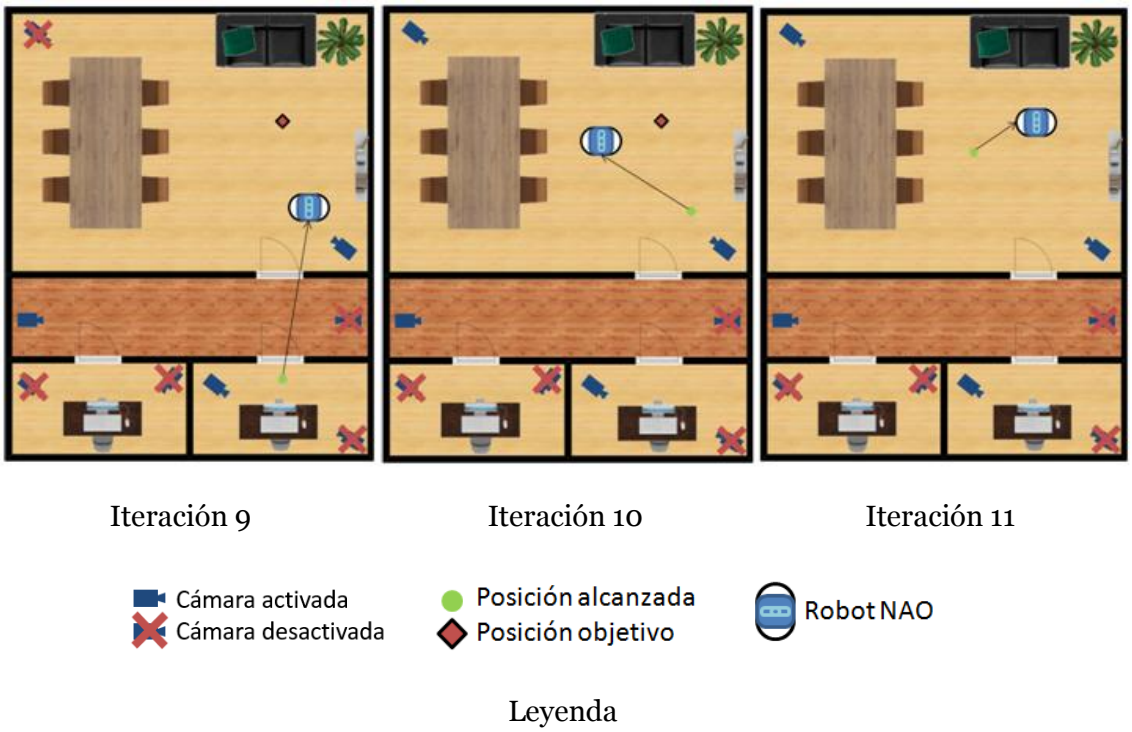


Figura 2 - Movimiento del robot NAO con corrección mediante cámaras

Una vez descrito el tipo de escenario donde se enmarca la solución que se propone en este trabajo, se procede a describir los subobjetivos que nos permitirán alcanzar el objetivo inicialmente planteado:

Estudiar diferentes herramientas y características de los agentes inteligentes y sistemas multiagente. Existe una gran variedad de plataformas multiagente con las que se puede trabajar, por lo que se realizará un estudio de estas plataformas en el capítulo 2 para determinar lo más adecuado para nuestro trabajo.

Estudiar las características y especificaciones del robot NAO. Para poder controlar el robot NAO, primero se deberá realizar una exhaustiva investigación sobre el robot.

Diseñar el sistema multiagente. Se deberá crear el comportamiento propio de cada uno de los agentes integrantes del sistema multiagente, así como la comunicación con estos.

Diseñar un sistema de notificación automática de presencia. Se pretende utilizar el concepto de notificación de presencia para conocer siempre el estado de los dispositivos de sensorización y así obtener un sistema multiagente transparente, permitiendo la incorporación de nuevos dispositivos de sensorización o el abandono de los ya incorporados.

Realizar diferentes pruebas del sistema multiagente diseñado y evaluar los resultados. Se realizarán diferentes pruebas de recorrido del

robot, siguiendo diferentes comportamientos de los agentes. Además, se analizarán los resultados obtenidos por el movimiento del robot.

1.3. Estructura del trabajo

La memoria de este trabajo contiene dos partes. La primera parte, consistirá en el estado del arte, compuesto por los dos primeros capítulos, donde se explicará detalladamente los conocimientos previos para la realización de este proyecto. En el primer capítulo, tratará sobre los sistemas multiagente, donde se explicará qué son los agentes inteligentes, así como sus características principales. También, se explicará el concepto de sistema multiagente, las características de estos, los diferentes tipos de características existentes y diferentes plataformas utilizadas. El último punto, versará sobre la historia de la robótica, explicando los diferentes tipos de robots móviles existentes y las características del robot NAO.

Por otro lado, en la segunda parte, se abordarán los cuatro últimos capítulos, donde se desarrollará la propuesta. El cuarto capítulo se corresponderá al diseño del escenario de pruebas, así como los recorridos que se realizarán en las pruebas. En el siguiente capítulo, se comentará el desarrollo del sistema multiagente. Seguidamente, en el sexto, se explicarán las diferentes pruebas y los resultados obtenidos. Finalmente, el último capítulo estará dedicado a comentar las conclusiones finales de este proyecto, junto a las líneas futuras de actuación.

Estado del arte

2. Sistemas multiagente

A lo largo de la historia se ha intentado definir el significado de los agentes inteligentes. Una de las definiciones que se puede encontrar es la definición de un agente inteligente como una entidad de software que, basándose en su propio conocimiento, realiza un conjunto de acciones para satisfacer a las necesidades de un usuario o de un programa, ya sea por iniciativa propia o porque alguno lo requiere [8]. También se puede encontrar la definición de un agente inteligente como una pieza de software que ejecuta una tarea utilizando información recopilada de un entorno posiblemente cambiante, por lo que este software debe auto-ajustarse, si el entorno así lo precisa, para poder completar la tarea [9].

Tras estas definiciones podemos llegar a la definición de que los agentes inteligentes sirven para ayudar a completar tareas definidas. Además, para completar estas tareas se basan en su conocimiento, que puede ser obtenido del entorno mediante sensores. Por último, a través de diferentes acciones, para cumplir la tarea asignada, puede modificar el entorno, por lo que deberá adaptarse. En la figura 3 se puede observar una representación gráfica de esta definición de agente inteligente.

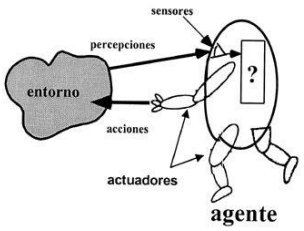


Figura 3 - Agente Inteligente

Los agentes inteligentes tienen las siguientes características:

- **Autonomía**: El agente tiene que realizar una tarea asignada, pero tiene el absoluto control de sus actos para poder resolverla.
- **Reactividad**: Los agentes pueden ser capaces de detectar los cambios de su entorno, y reaccionar dependiendo de estos cambios.
- **Adaptabilidad**: Esta característica está ligada a la reactividad, y se basa en la adaptación del agente al entorno.
- **Sociabilidad**: El agente tiene la capacidad de comunicarse con las personas o con otros agentes. Esta característica podría utilizarse para la cooperación entre otros agentes.
- **Pro-actividad**: Esta característica se basa en que los agentes tengan la iniciativa para resolver los problemas o tareas asignadas.
- **Continuidad temporal**: Los agentes deben estar en disposición continua, debido a que desde su creación esperan a recibir algún cambio del entorno o algún evento provocado por el usuario u otros agentes.

Los sistemas multiagente (SMA) son sistemas compuestos por dos o más agentes inteligentes que actúan sobre el mismo entorno. La acción de un agente sobre un entorno puede afectar de muchas maneras a los agentes que actúan sobre este entorno, por eso resulta interesante la idea de tener un SMA donde los agentes pueden tomar las decisiones según las decisiones tomadas por otros agentes.

La necesidad de estos sistemas viene dada por utilización de diferentes agentes para diferentes tareas o de las mismas tareas. La gran mayoría de las tareas pueden realizarse únicamente por un agente, pero la idea que se suele seguir es dividir las diferentes tareas en sub-tareas para poder utilizar diferentes agentes [10]. Esta idea tiene en mente dividir tareas complejas en sub-tareas más simples donde los datos están distribuidos entre los diferentes agentes. Al tener un SMA los agentes pueden cooperar para realizar sus tareas de manera más eficaz, ya que puede que necesiten información que solo dispone unos determinados agentes, como en el caso de este trabajo donde el agente Robot necesita la ayuda de diferentes agentes para conocer la posición se encuentra. Por otro lado, los SMAs no solo se utilizan para que los agentes cooperen, puede que estos no deseen cooperar, un claro ejemplo de esto lo vemos en los sistemas de subastas, donde cada agente actúa por su propio beneficio [11].

Los SMA suelen caracterizarse de la siguiente manera:

- La información del problema está dividida entre los agentes, por lo que los datos están descentralizados.
- Los agentes son sociales y pueden mandar mensajes a los diferentes agentes.
- Puede existir diferentes tipos de cooperación entre los agentes para que cada uno cumpla sus objetivos.
- Los agentes pueden actuar de manera veraz o engañosa, retransmitiendo información incorrecta, ya sea por su programación o por datos anómalos obtenidos.

Dicho esto, podemos enumerar diferentes ventajas que nos aporta utilizar un sistema distribuido para resolver las tareas:

- Al tener un sistema distribuido, se evitan los posibles problemas de limitación de recursos. También podríamos evitar perdida de información si un agente fallara, ya que la información de un agente puede estar contenida en varios agentes, y al fallar este no tendríamos problemas para resolver el problema.
- Permite la conexión entre los diferentes agentes, que actúan sobre el mismo entorno.
- Se obtienen soluciones eficientes a los problemas.
- Permite flexibilidad para realizar las tareas, ya que el número de agentes dentro del SMA puede ser dinámico.

Los agentes, dentro del SMA, pueden actuar de manera diferentes según su grado de cooperación. Básicamente podemos clasificarlos en tres tipos de agentes diferentes [12]:

- **Agentes hostiles**: Son aquellos agentes que realizan una tarea que masifica su valor de utilidad en función de perjudicar el valor de utilidad de las tareas de otros agentes. En el caso de un sistema de ventas tendríamos un agente cliente y otro agente vendedor, donde el cliente quiere adquirir los productos al menor coste mientras que el vendedor quiere venderlos al mayor coste, este sería un caso claro donde se podrían aplicar este tipo de agentes.
- **Agentes auto-interesados**: Estos agentes únicamente se interesan por realizar sus tareas para maximizar su propio beneficio, y no para maximizar el beneficio del SMA. Estos agentes suelen utilizarse en competiciones, donde varios agentes compiten en el mismo entorno.
- **Agentes colaborativos**: Estos son los agentes que trabajan juntos para realizar una tarea y así maximizar su beneficio común. Básicamente se basan en centrarse en el bien común independientemente del bien individual.

2.1. Tipos de mensajería

Como ya hemos comentado los SMAs permite que diferentes agentes situados en el mismo entorno puedan realizar tareas cooperando entre ellos. Para poder realizar esta cooperación, los agentes deben comunicarse utilizando el mismo lenguaje de comunicación. También es necesario determinar que estructura sigue el grupo de agentes. La técnica de coordinación más exitosa para la asignación de recursos y tareas entre una sociedad de agentes es *Contract-Net Protocol* (CNP) adoptado por en las especificaciones de *Foundation for Intelligent Physical Agents* (FIPA). Utilizando la técnica de CNP podemos aplicar dos roles distintos a los agentes, director y contratista [13]. Esta técnica de coordinación divide el problema asignado a un agente en diferentes sub-problemas, si este no puede resolverlo. Entonces el agente en cuestión se convierte en director y difunde los nuevos sub-problemas a los diferentes agentes (contratistas), los cuales pueden aceptar o no el sub-problema.

La FIPA es un organismo para el establecimiento de estándares de software para sistemas basados en agentes. Fundada como una asociación sin ánimo de lucro en 1996, FIPA buscaba definir un conjunto de normas para la implementación de sistemas en los que se pudieran ejecutar agentes [13]. Las plataformas que deseen seguir este modelo deben cumplir tres características:

- **ACC** (Agent Comunnication Chanel): Se trata de un canal de comunicación que permite la comunicación entre los agentes del sistema y de la plataforma.
- **AMS** (Agent Management System): Se registra los agentes en la plataforma para que todos los agentes registrados tengan el contacto (páginas blancas).
- **DF** (Directory Facilator): Parecido al AMS debemos hacer un registro de todos los servicios que ofrece cada uno de los agentes (páginas amarillas).

Una vez explicada la necesidad de la comunicación entre agentes se va a realizar una pequeña revisión de algunos de los estándares de comunicación existentes entre los agentes.

Agent Communication Language

El *Agent Communication Language* (ACL) es un estándar de lenguaje propuesto por FIPA. Este lenguaje posiblemente sea el más conocido en el mundo de los sistemas basados en agentes [14, 15, 16, 17, 18, 19, 20]. Este lenguaje está compuesto por dos tipos de lenguajes distintos:

- **Knowledge Interchange Format** (KIF): Lenguaje basado en la lógica de predicado de primer orden. Los agentes que utilizan este tipo de lenguaje pueden expresar características de objetos, relaciones entre diferentes objetos, etc.
- **Knowledge Query and Manipulation Language** (KQML): Lenguaje que permite manipular el conocimiento. Los mensajes retransmitidos tienen la información parametrizadas dentro de diferentes clases.

Mensajería instantánea

La mensajería instantánea se basa en la comunicación entre dos o más agentes en tiempo real, donde el mensaje retransmitido es únicamente texto escrito, sin ningún tipo de codificación. Cabe destacar que antes de realizar este tipo de mensaje, el emisor conoce si está disponible el receptor. La mensajería instantánea permite una rápida comunicación entre las diferentes partes que intervienen en dicho mensaje.

XMPP/Jabber

Extensible Messaging and Presence Protocol (XMPP), originalmente llamado *Jabber*, es un protocolo de mensajería instantánea basada en XML diseñado en 1999 por Jeremie Miller. Este protocolo contiene notificaciones de presencia, por lo que un agente recibe información sobre la presencia de otros agentes que se encuentran en el mismo sistema. Entre sus principales características tenemos que se trata de un protocolo abierto y libre, asincrónico, descentralizado, seguro, flexible y multiusuario.

2.2. Plataformas de sistemas multiagente

En este apartado se van a presentar alguna de las plataformas que se utilizan para los SMA. Existen diversas plataformas, en este apartado se explicará las características de tres plataformas, una de estas será la plataforma utilizada en el trabajo realizado, concretamente la plataforma *SPADE*.

Multi-Agent Development Kit (MADKit)

MADKit es una plataforma multiagente escrita en Java, bajo los términos de licencia de GNU GPL. Esta plataforma fue propuesta por *Jacques Ferber* y desarrollada en el Laboratorio de Informática, Robótica y Microelectrónica de Montpellier (LIRMM). Esta plataforma es interesante debido a que no tiene una estructura interna de los agentes, permitiéndonos desarrollar libremente nuestras arquitecturas, creando y gestionando el ciclo de vida de nuestros agentes [21, 22].

En MADKit está orientada para desarrollar SMAs basadas en un paradigma orientado a la organización, por lo que podemos definir al agente como una entidad que se comunica y posee diferentes roles dentro de un grupo. Estos grupos representan diferentes agregaciones de agentes, mientras que los rolos representan la función del agente, los servicios que ofrece o su identificación dentro del grupo. En la figura 4 se puede observar de forma visual la relación entre el agente, los grupos y los rolos. Además, una gran ventaja que tiene esta plataforma es su fácil distribución. Los agentes se pueden distribuir sin necesidad de cambiar nada en el código del agente [23, 24].

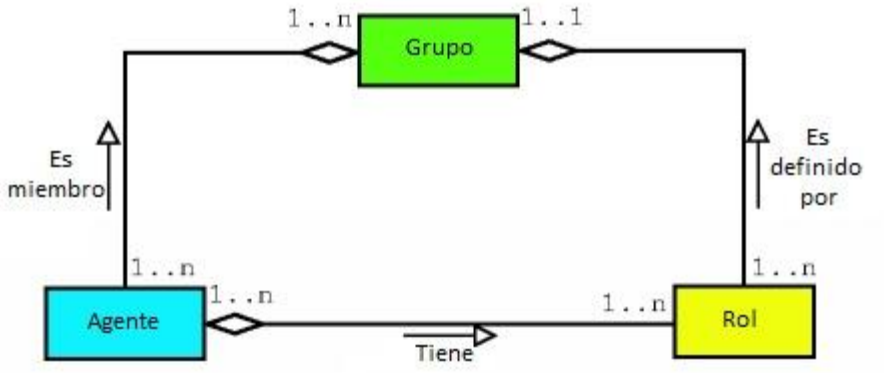


Figura 4 - Relación entre los agentes, los grupos y los roles en *MADKit*

Java Agent Development Framework (JADE)

JADE es un software libre, bajo los términos de licencia de LGPL, implementado en Java. Este software fue desarrollado por la empresa *Telecom Italia* en el 2000. La idea principal que seguía esta plataforma es permitir la implementación de agentes de manera simplificada siguiendo los estándares establecidos por FIPA, además de utilizar el lenguaje de programación FIPA ACL. Por otro lado, la plataforma puede ser distribuida en varios hosts, y podemos controlar la configuración del SMA mediante una interfaz gráfica remota, esta característica ayuda en gran medida de tener un control eficiente de un SMA distribuido [25, 26].

Como JADE sigue las especificaciones establecidas por FIPA, podemos encontrar que disponemos de dos agentes especiales:

- **Agente DF**: Agente que informa sobre qué agentes están disponibles en el SMA.
- **Agentes AMD**: Tiene el control sobre la creación y destrucción de los agentes, e incluso puede detener la plataforma.

Además, los agentes definidos en esta plataforma deben seguir el ciclo de la vida propuesto también por FIPA, donde un agente puede tener hasta seis estados diferentes [27]:

- **Iniciado**: El agente ha sido creado, pero aún no ha sido registrado.
- **Activo**: Ha sido registrado y puede comunicarse con otros agentes.
- **Suspendido**: Su tarea a realizar ha sido suspendida.
- **Esperando**: Espera el suceso de un evento.
- **Eliminado**: El AMS ha eliminado al agente.
- **Transito**: El agente está migrando a una nueva ubicación.

En la figura 5 se puede observar los posibles estados de un agente y la manera que se pueden transitar a cada uno de ellos de manera gráfica.

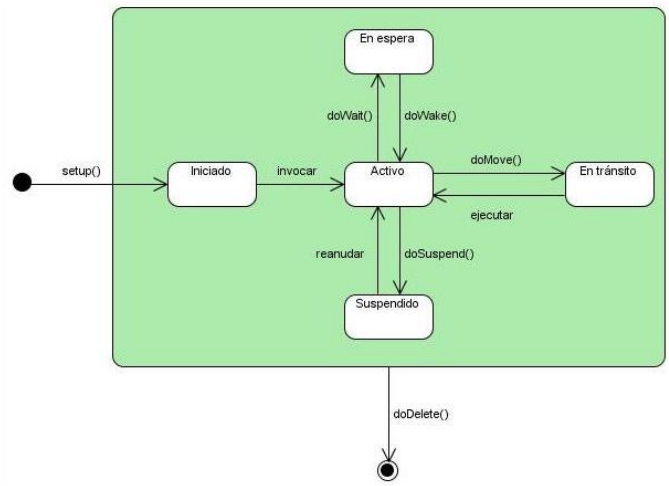


Figura 5 - Visión gráfica de los estados de un agente en *JADE*

Smart Python multi-Agent Development Environment (SPADE)

SPADE es una plataforma libre de SMA desarrollada en 2005 en la Universidad Politécnica de Valencia por J. Palanca y G. Aranda [28, 29]. A diferencia de las otras plataformas, *SPADE* es la primera que se basa en mensajería instantánea XMPP, explicado en el apartado 3.2.1, además está desarrollada con el lenguaje de programación Python. Esta plataforma se basa en diferentes conjuntos de estándares como FIPA y XMPP/Jabber [28].

Las principales características que podemos encontrar en la plataforma *SPADE* son las siguientes [29]:

- Al dar soporte al estándar FIPA, admite las características de AMS (páginas blancas) y DF (páginas amarillas).
- Permite la conversación a más de un agente gracias al mecanismo Jabber.
- Es posible utilizar la notificación de presencia permitiendo al sistema y a los diferentes agentes conocer en tiempo real quien está conectado.
- Los agentes creados necesitan un usuario y una contraseña, por lo que ayuda a mejorar la seguridad del sistema, además permite una conexión cifrada.
- Permite la comunicación P2P entre los agentes.
- Es posible definir modelos de agente BDI basados en creencia (*Belief*), deseos (*Desire*) e intenciones (*Intention*).
- Se puede crear diferentes comportamientos para un agente (cíclicos, periódicos, *timeout*, máquina de estados infinitos, etc). Además, un agente puede realizar diferentes tareas.
- Da soporte de comunicación con otras plataformas siguiendo diversos protocolos de transporte como HTTP o XMPP.
- Mediante el estándar de PubSub permite la publicación o subscripción de eventos.
- Dispone de una interfaz gráfica en la web que nos permite visualizar los agentes que están conectados y la disponibilidad de eventos entre otras.

En la figura 6 se puede observar un pequeño esquema de la arquitectura de *SPADE*, donde aparece alguna de las características mencionadas.

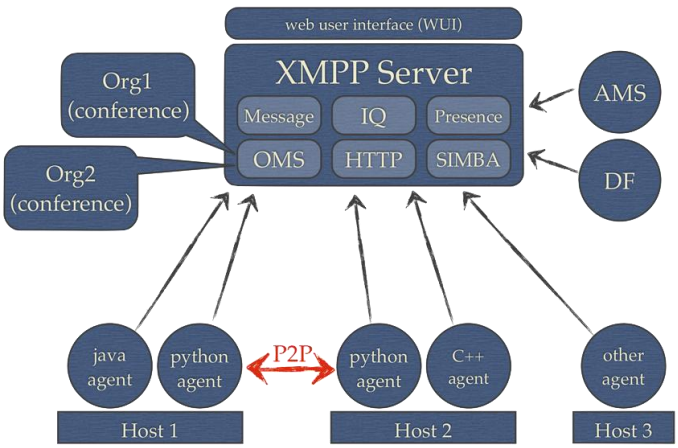


Figura 6 - Arquitectura *SPADE*

Por último, mencionar que en este trabajo se ha utilizado esta plataforma, debido a que vistas sus características observamos que puede ser interesante aplicarla a este estudio. Además, como está diseñada en Python, facilita la interacción con los diferentes dispositivos de sensorización utilizados y la integración con el robot.

3. Robótica

En los últimos años la robótica y la inteligencia artificial han ido cogiendo popularidad e interés por la población. El ser humano ha estado diseñando diferentes artefactos con el fin de mejorar la producción, y mejorar la eficiencia de sus mercados, e incluso han desarrollado diferentes robots humanoides que intentan simular a las personas. Además del diseño de estos artefactos o robots se ha intentado aplicarles una inteligencia para que puedan operar de manera autónoma, sin la necesidad de un control constante.

La idea de robot apareció por primera vez una obra teatral de ciencia ficción llamada *Rossum's Universal Robots* en 1920. Su autor, Karel Čapek, baso su novela en una empresa que creaba humanoides para ayudar a los seres humanos, donde más adelante entran en conflicto con estos destruyendo la humanidad.

Unos años más tarde, la Corporación Eléctrica Westinghouse creo el primer robot humanoide de la historia, en 1939. El robot, que se puede observar en la figura 7, tenía la capacidad de decir alrededor de 700 palabras, fumar cigarrillos e incluso caminar tras decirle un comando de voz.

Figura 7 - Primer robot humanoide

Más tarde, Isaac Asimov publicó un libro llamado Runaround en 1942, donde se enunciaban las tres leyes de la robótica [30]:

1. Un robot no hará daño al ser humano, ni permitirá que el ser humano sufra daño.
2. Un robot debe obedecer las órdenes del ser humano, siempre que no entren en conflicto con la primera ley.
3. Un robot debe proteger su existencia siempre que no entre en conflicto con la primera o segunda ley.

Existen una gran variedad de robots, pero en este trabajo únicamente nos vamos a centrar en presentar los diferentes tipos de robots que existen dentro de la robótica móvil [31].

3.1. Robots móviles

La peculiaridad de este tipo de robots es que pueden moverse por el entorno, por lo que hace interesante utilizarlos para realizar diferentes experimentos. Concretamente vamos a diferenciar cinco tipos de robots diferentes: rodadores, reptadores, nadadores, voladores y andantes.

Robots rodantes

Este tipo de robots se desplazan por el entorno a través de unas ruedas o cadenas que tienen incorporadas, existen con mucha variedad de robots de este estilo, debido a que son los más sencillos de diseñar. La manera más sencilla de diseñar este tipo de robot es utilizar dos ruedas, una en cada lado del robot, y una rueda loca en el centro. En la figura 8 podemos observar a *Mars Exploration Rover*, cuyo diseño hace que este dentro de los robots rodantes, ya que dispone de seis ruedas para su desplazamiento.

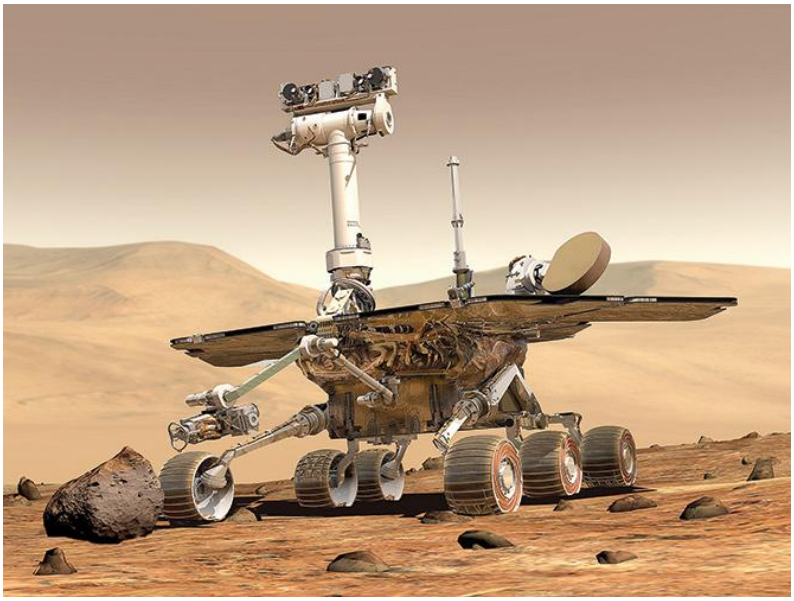


Figura 8 - Robot rodante

Robots reptadores

Los movimientos de este tipo de robots se basan a la mayoría de los animales reptadores, como las serpientes o los gusanos. Este tipo de robots están compuesto por diferentes secciones que realizan diferentes movimientos siguiendo una coordinación que permite que se deslicen por el suelo. En la figura 9 se puede observar un robot reptador, el comportamiento de este robot se basa en la naturaleza de las serpientes.

Figura 9 - Robot reptador

Robots nadadores

Este tipo de robots están adaptados especialmente para moverse en el agua. Suelen estar orientados para realizar exploraciones submarinas, tareas de mantenimiento subacuáticas y como herramienta de rescate. En la figura 10 se puede observar un robot nadador, además este robot también entraría dentro de la sección de robot reptador, ya que se basa en la naturaleza de las serpientes y puede moverse por suelo firme.

Figura 10 - Robot nadador

Robots voladores

Este tipo de robots tienen la capacidad de desplazarse por el aire gracias a una serie de hélices que tienen incorporadas, que generar la suficiente fuerza para poder levantar el peso del robot. En la figura 11 se puede observar un robot volador que dispone de 4 hélices para moverse por el entorno.

Figura 11 - Robot volador

Robots andantes

Por último, los robots andantes, o humanoides, se diferencian a los demás por su capacidad de simular los movimientos del ser humano para desplazarse por el entorno. Este tipo de robots se componen principalmente de dos piernas y va manteniendo el equilibrio mientras se desplaza. Estos tipos de robots son muy inestables ya que es muy difícil para los robots mantener el equilibrio, pero gracias al gran avance de las tecnologías se ha conseguido que este tipo de robots corran, bailen, gateen e incluso que practiquen diferentes deportes. En la figura 12 podemos observar un robot humanoide, concretamente el robot NAO que se utilizará en este trabajo.

Figura 12 - Robot andante

3.2. Robot NAO

El robot NAO fue creada por la empresa Aldebaran Robotics en Francia en 2008 y dispone de varios sensores (sonar, micrófonos, cámaras HD, sensores táctiles, acelerómetros, wifi, etc) que hacen que sea muy interesante utilizarlo para diferentes experimentos. Este robot ha conseguido bastante éxito y se ha convertido en el robot más utilizado en la Robocup. En la figura 13 se puede observar la gran variedad de sensores que dispone este robot.

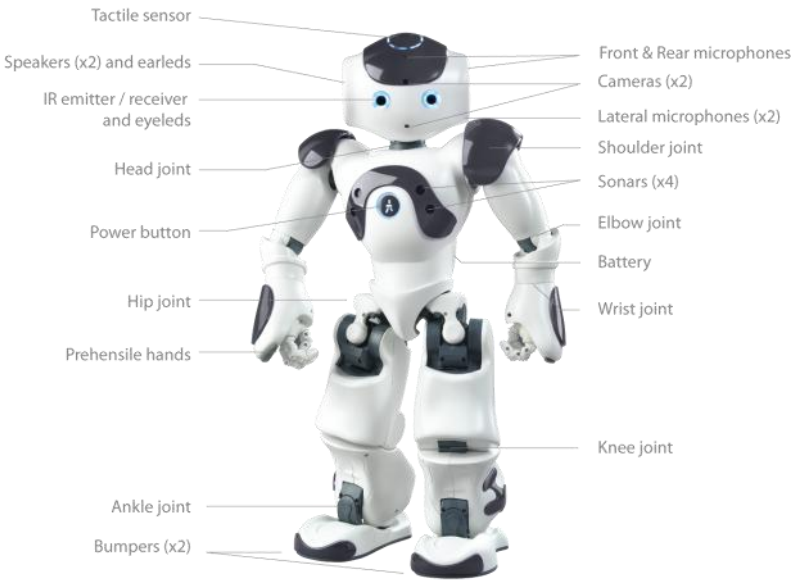


Figura 13 - Sensores del robot NAO

La estructura del robot se puede descomponer en cinco partes [32]:

- **Head**: Se trata de la cabeza del robot. Esta tiene la posibilidad de moverse de izquierda a derecha y de arriba abajo.
- **LArm/RArm**: Estas partes corresponden a son el brazo izquierdo y derecho respectivamente. Permiten realizar todos los movimientos del brazo, tanto del hombro como del codo.
- **LHand/RHand**: Se trata de la mano izquierda y la mano derecha respectivamente. Estas tienen la posibilidad de abrir o cerrar la mano, así como girar la muñeca.
- **LLeg/RLeg**: Estas partes corresponden a la pierna izquierda y derecha respectivamente. Permiten subir o bajar las piernas, mover las rodillas y mover los tobillos.
- **Torso**: Por último, tenemos el torso del robot donde está el acelerómetro, los dos sonares y un giroscopio. Los sensores del torso permiten al robot mantener el equilibrio, así como prevenir los golpes.

Los robots NAO pueden ser programados en diferentes lenguajes de programación: Python, C++, Java, MATLAB, etc. En este proyecto vamos a utilizar el lenguaje de programación de Python, debido a que es el mismo lenguaje de programación que utilizamos con la plataforma SMA, *SPADE*.

Aldebaran Robotics ha introducido un sistema operativo basado en Linux llamado NAOqi a los robots NAO. Además, esta organización ha diseñado un simulador llamado *Choregraphe* con el que podemos programar el robot y simular sus movimientos. Aparte de este simulador existen otros como por ejemplo Webots, que es utilizado para simular el comportamiento de una gran cantidad de robots programables.

NaoQi

El sistema operativo que tiene incorporado el robot NAO es NaoQi. Este sistema operativo está basado en Linux. Aldebaran Robotics creó el *framework* NaoQi, el cual nos permite programar al robot a través de un ordenador, es decir, el robot NAO posee este *framework* como sistema principal, y mediante un ordenador al que también tenemos instalado este *framework* podemos controlar el funcionamiento del robot y controlar los valores captados por sus sensores [32]. Este *framework* dispone de seis módulos con los que podemos interactuar para controlar al robot:

- **NaoQiCore**: Relacionados con la memoria del robot y los ficheros de configuración.
- **NaoQiVision**: Permiten utilizar las cámaras del robot, identificando caras, realizando videos o capturas.
- **NaoQiSensors**: Ofrecen la información de los sensores del robot.
- **NaoQiTrackers**: Ofrecen funciones para seguir diferentes objetos, caras o marcas.
- **NaoQiMotion**: Permiten desplazar al robot, revisar el estado de sus articulaciones y prever choques o caídas.
- **NaoQiAudio**: Permiten reconocimiento de voz, reproducir diferentes sonidos e interactuar con los usuarios.

Simulador Choregraphe

La organización Aldebaran Robotics ofrece el software *Choregraphe* para la programación y simulación del robot NAO. Con este software se puede ejecutar distintos comandos del robot desde un ordenador remoto. Estos comandos se pueden organizar mediante un diagrama de bloques para controlar el comportamiento del robot, donde cada bloque representa una parte del comportamiento del robot [32]. En la figura 14 se puede observar la interfaz de este software y se puede encontrar un ejemplo del diagrama de bloques anteriormente comentado. En la derecha de la figura se puede observar la simulación del robot, mientras que en la parte de la izquierda

podemos elegir diferentes bloques ya totalmente implementados para añadirlos a nuestro robot.

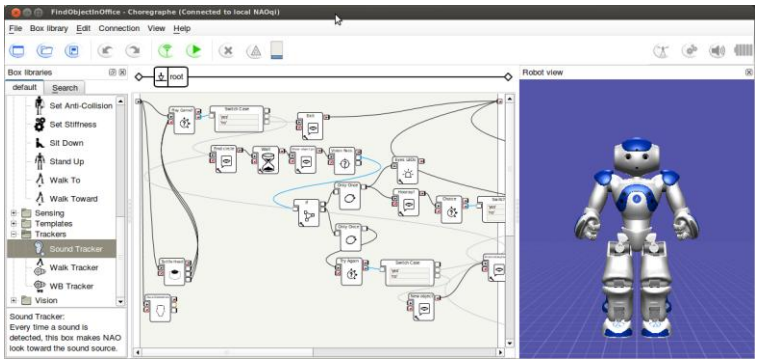


Figura 14 - Interfaz *Choregraphe*

Choregraphe es un software muy intuitivo. Además, Aldebaran Robotics ha elaborado una guía para ayudar para los usuarios que acaban de empezar a usar el software.

Simulador Webots

Se trata de un software utilizado para simular robots móviles. Este simulador fue desarrollado por *Cyberbotics* y permite crear un entorno y programar un robot para que actúe sobre él. Lo interesante de este simulador es la posibilidad de utilizar diferentes escenarios, ya sea cargándolo o creándolo [33]. En la figura 15 se puede observar la interfaz de este software. En la derecha de la figura se puede escribir el código correspondiente a la programación del robot. Por otro lado, en el centro de la figura se puede observar cómo funciona la simulación. Por último, a la izquierda se puede observar el árbol de escena, que muestra todos los elementos que contiene el entorno. Su facilidad de simular los escenarios, hace que sea un software muy utilizado para la educación.

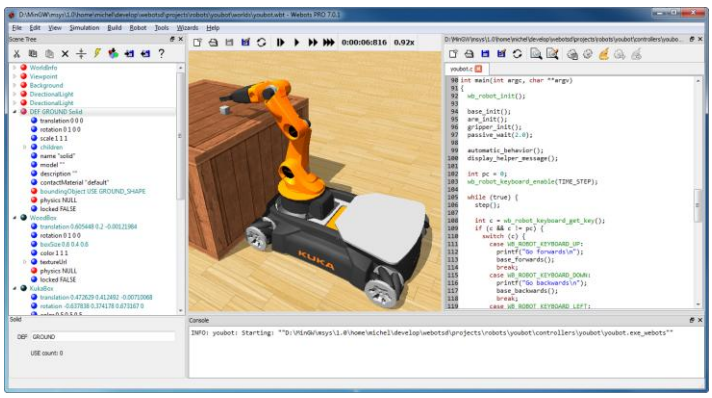


Figura 15 - Interfaz *Webots*

Propuesta

4. Diseño del entorno

En este capítulo, se va a proceder al desarrollo del diseño del entorno, en el cual se va a desplazar el robot NAO, así como la descripción de los dispositivos de sensorización que se van a emplear y los diferentes recorridos que se utilizarán en las pruebas.

Una de las causas del error del movimiento del robot NAO era el pavimento por el que se desplazaba. Por ese motivo, se probó el movimiento del robot sobre mármol, césped artificial y madera. Tras realizar un movimiento simple del robot, se pudo comprobar que el césped artificial ejercía una mayor fuerza de rozamiento que las anteriores, causando que el robot obtuviera una menor precisión. Por otro lado, utilizando mármol el robot se deslizaba en exceso sobre el suelo obteniendo unos resultados levemente mejores que el césped. Por último, se probó sobre la madera, y como se alcanzó una posición más precisa que las obtenidas con los anteriores pavimentos mencionados, se decidió utilizar un suelo de madera en el entorno. Dicho esto, para poder realizar las pruebas necesarias que comprobarán el funcionamiento del sistema multiagente desarrollado en este trabajo, se decidió dividir el escenario en 36 cuadrículas (9 verticales por 4 horizontales). Una vez realizada dicha división, se podía fácilmente distinguir en qué cuadrícula se encontraba el robot. Cada cuadrícula tenía un tamaño de 625 cm^2 (25x25 cm^2), por lo que el escenario tenía un tamaño de 2,25 m^2. En la figura 16 se puede observar el escenario de pruebas utilizado.

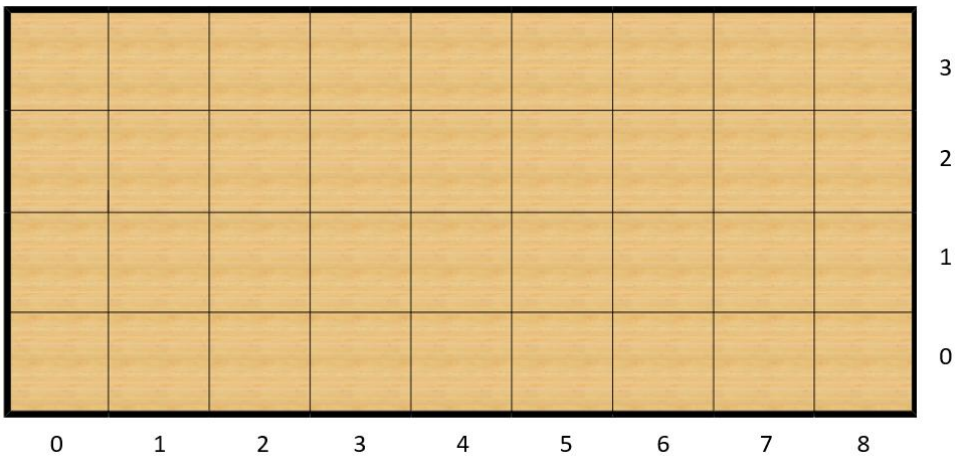


Figura 16 – Escenario de pruebas

Siguiendo con el diseño del entorno, se han utilizado diferentes dispositivos de sensorización en este trabajo, en concreto una *Xtion PRO LIVE* y cuatro *Raspberry Pi* 2 con cámaras integradas. Para conocer la posición real del robot, se necesita obtener la posición en los ejes x e y. La *Xtion PRO LIVE* posee sensores de profundidad e infrarrojos que permiten detectar la profundidad de los objetos, muy útil para identificar la posición del robot, ya que puede obtener los dos ejes. Por otro lado, las *Raspberry Pi* 2 con cámaras integradas no detectan la profundidad, es decir,

únicamente detectan un eje, por lo que complican el cálculo de la posición del robot. En la figura 17 se puede observar la apariencia de estos dispositivos.

Raspberry Pi 2 con cámara integrada *Xtion PRO LIVE*

Figura 17 – Dispositivos de sensorización utilizados

Además de emplear estos dispositivos de sensorización, se ha utilizado la librería *OpenCV* disponible para *Python* (y otros lenguajes de programación como *C++*, *C* o *Java*), que permite tratar las imágenes captadas por los dispositivos de sensorización y, por tanto, trabajar con las imágenes obtenidas por la *Xtion PRO LIVE*. Gracias a esta librería, se han conseguido crear diferentes algoritmos para los dispositivos de sensorización, que devuelven la posición real del robot, y que se explicarán más adelante en la sección 5.2. La figura 18 hace referencia al entorno físico utilizado y el robot sobre éste.

Figura 18 – Entorno físico utilizado

4.1. Diseño de las pruebas

Una vez decidido el pavimento que se debía utilizar, y haber seleccionado y programado los dispositivos de sensorización, para realizar las pruebas, se decidió utilizar tres tipos de recorridos distintos con el fin de probar el sistema multiagente desarrollado. Estos recorridos son los siguientes:

- **Recorrido circular**: El robot intenta realizar un movimiento circular. Resulta un recorrido sencillo, con giros de pocos grados. En la figura 19 se puede apreciar el recorrido.

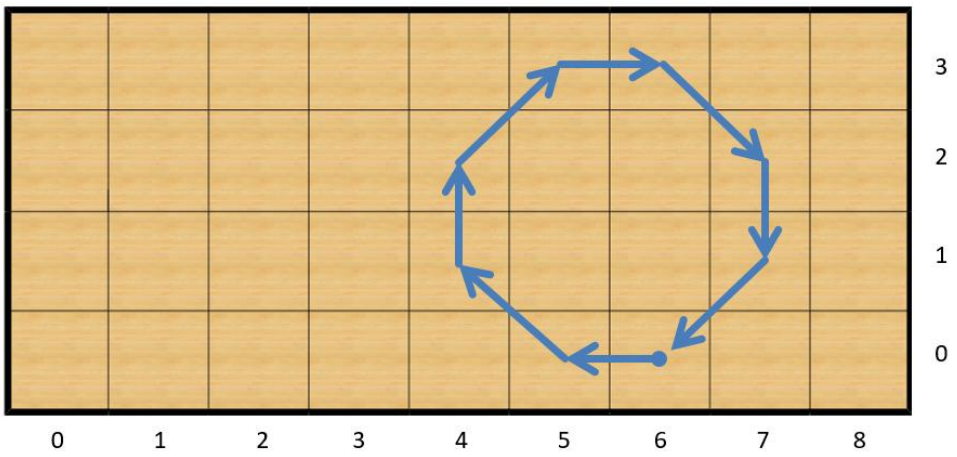


Figura 19 – Recorrido circular

- **Recorrido estrella pitagórica**: Como su nombre indica, el robot intenta dibujar mediante su recorrido una estrella de cinco puntas. Se decidió utilizar este recorrido debido a la cantidad de giros que debía dar el robot para realizar la estrella. Todos los giros que se realizan en este recorrido son mayores a 90º. La figura 20 se corresponde al recorrido descrito.

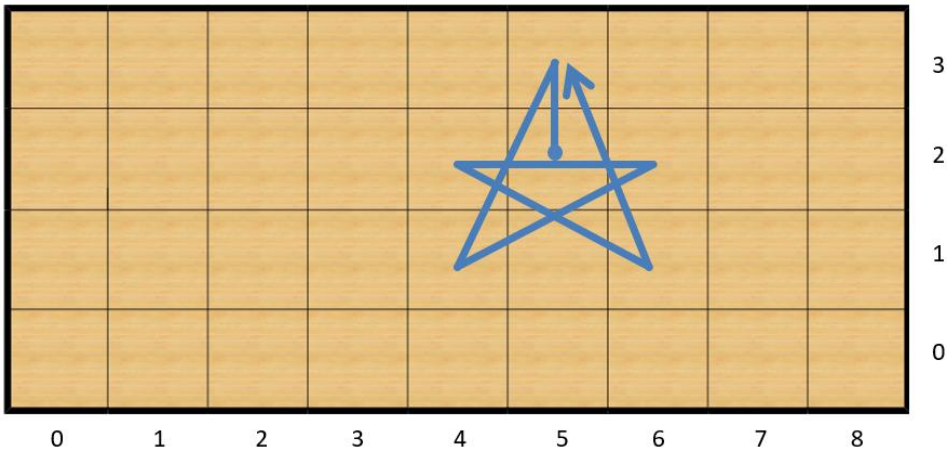


Figura 20 – Recorrido estrella pitagórica

- **Recorrido zigzag**: Debido a que los otros recorridos, son recorridos con pocos movimientos, se decidió utilizar un recorrido más largo. Este recorrido acaba donde empieza, por eso se decidió realizar dos vueltas. En la figura 21 se observa el recorrido.

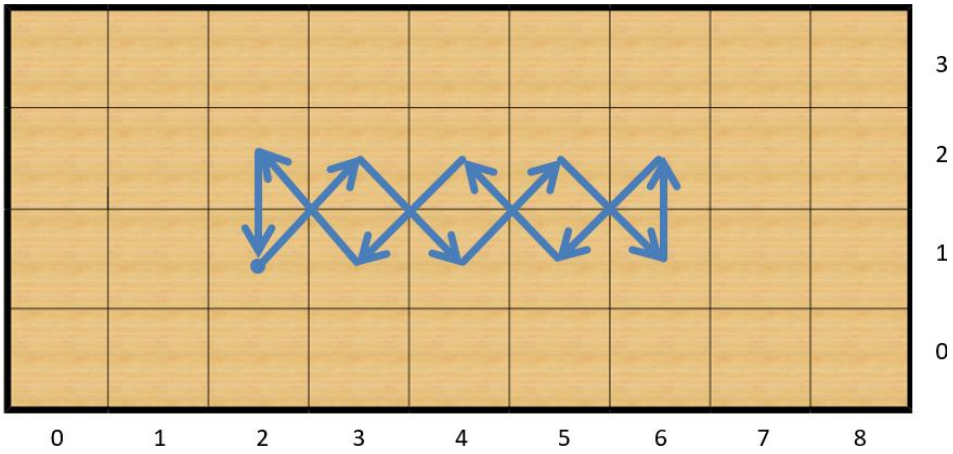


Figura 21 – Recorrido zigzag

5. Diseño del sistema multiagente

Tal y como se planteó en el capítulo 1, el objetivo principal de este trabajo es ayudar a alcanzar la posición deseada por el robot al desplazarse por diferentes zonas o salas. Para cumplir este objetivo, se ha utilizado la plataforma *SPADE* explicada en la sección 3.2. Gracias a esta plataforma podemos obtener una solución flexible y distribuida que nos permite que diferentes dispositivos de sensorización se incorporen o salgan del SMA de manera transparente. Esta plataforma nos sirve de puente entre los diferentes agentes que controlan los dispositivos de sensorización, que a partir de ahora se llamarán agentes dispositivos, y el agente que controla el robot, que se nombrará como agente robot, tal y como podemos observar en la figura 22.

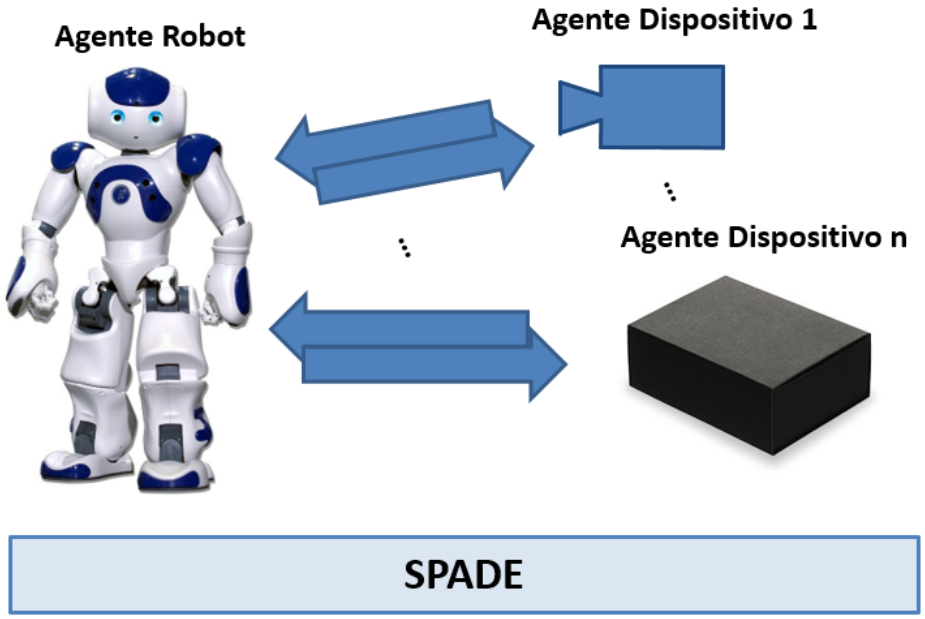


Figura 22 – *SPADE* como puente entre los agentes

De esta forma, los agentes que se deben diseñar para incorporar al SMA desarrollado en este trabajo son dos: el agente robot y el agente dispositivo que contiene la información sobre la posición del robot. Este último agente puede aplicarse a distintos dispositivos de sensorización únicamente modificando su cálculo de la posición del robot.

5.1. Agente robot

El comportamiento del agente robot consiste en tratar la información recibida por distintos agentes y manejar al robot. La figura 23 hace referencia a un diagrama que muestra el conjunto de comportamientos del robot, seguido de la explicación de cada componente de su comportamiento.

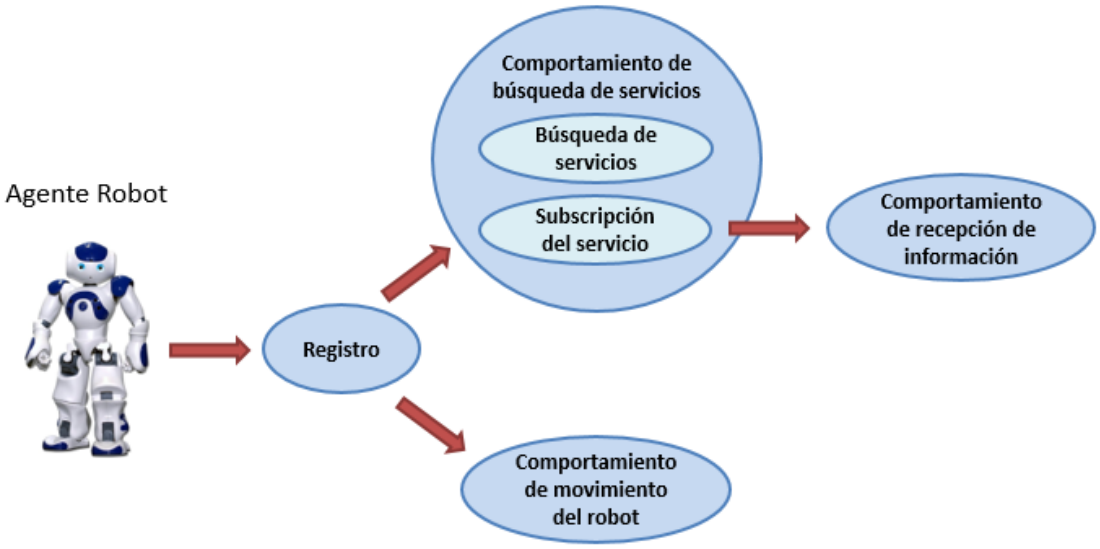


Figura 23 – Comportamientos seguidos por el robot

Registro

Nada más iniciarse, el agente deberá registrarse dentro del sistema multiagente a través del AMS. Para poder registrarse debe seguir los pasos propuestos por la plataforma *SPADE*: asignarle un identificador llamado *Jabber ID* (JID), seguido del *host* en el que se esté ejecutando el sistema multiagente, y una contraseña con la que conectarse. Cabe destacar que cada agente deberá tener un identificador JID diferente. La figura 24 muestra el registro del agente robot, que se resume en una única línea.

```
a = MyAgent("agente_robot@host", "contraseña")
```

Figura 24 – Registro del agente robot en la plataforma *SPADE*

Búsqueda de servicios

Los agentes dispositivos van a publicar los servicios de localización que ofrecen como servicios del DF. Por ese motivo, el agente robot necesita realizar una búsqueda de los servicios, a través del DF, de manera iterativa. Para llevar a cabo esta búsqueda, se le asigna un comportamiento cíclico, siguiendo la estructura existente de comportamientos cíclicos de la plataforma *SPADE*, que se llama "comportamiento de búsqueda de servicios". En este comportamiento, se realizará la búsqueda de servicios como se muestra en la figura 25, donde primero se crea la descripción de un agente, al que se le asigna un servicio del tipo que se quiere buscar, y, después, se realiza la búsqueda obteniendo todos los agentes que ofrecen ese servicio.

```
dad     = spade.DF.DfAgentDescription()
service = spade.DF.ServiceDescription()
service.setName("Camera")
dad.addService(service)
result  = myAgent.searchService(dad)
```

Figura 25 – Búsqueda de servicios por el agente robot

Subscripción del servicio

En este trabajo se va a hacer uso del protocolo de publicación y subscripción de *SPADE*, el cual permite a un agente crear un evento o subscribirse a los eventos creados por otros agentes a los que se está subscrito a través de la notificación de presencia. Con esta subscripción, el agente subscriptor queda a la espera de nuevas noticias ofrecidas por el agente publicador, que puede enviar mensajes en cualquier momento a todos aquellos agentes que estén subscriptos al evento. Dicho esto, los servicios ofrecidos por los agentes dispositivos van a utilizar este protocolo de publicación y subscripción, siendo éstos los publicadores. Por tanto, después de realizar la búsqueda de los diferentes servicios, dentro del comportamiento de búsqueda de servicios, el agente robot debe tomar el papel de subscriptor, subscribiéndose a todos aquellos servicios ofrecidos por los agentes que ha devuelto la búsqueda. Para realizar esta subscripción, primero se debe subscribir al agente en cuestión, para hacer uso de la notificación de presencia, y después se debe subscribir al servicio ofrecido añadiendo un comportamiento para tratar la información del servicio. En la figura 26 se puede observar cómo se realizaría la subscripción en la plataforma *SPADE*. Cabe destacar que, al hacer uso de la notificación de presencia, se puede conocer el estado de las cámaras, conociendo cuándo se desconectan o conectan al sistema multiagente.

```
myAgent.roster.subscribe( agent )
myAgent.subscribeToEvent( EventName, myAgent.ReceiveBehav )
```

Figura 26 – Subscripción de servicios por el agente robot

Recepción de información de la subscripción

Gracias a la subscripción del servicio, el agente que lo está ofreciendo mandará el mensaje al agente robot, por lo que este último deberá prepararse para recibirlos y tratarlos como es debido. Para tratar estos mensajes se utilizará un comportamiento basado en eventos que se llama "comportamiento de recepción de información". Este comportamiento se ejecutará únicamente cuando el robot reciba información de alguna de sus subscripciones y almacenará la información recibida junto al nombre del agente que la ha mandado y la hora de la recepción. Si ya tenía información guardada de ese agente se sobrescribe. La figura 27 se corresponde a este comportamiento, que se ejecuta cada vez que se recibe un mensaje.

```
Algoritmo RecepcionInformacion
        Posicion <- Recibir()
        Escribir Agente <- Posicion, Tiempo
FinAlgoritmo
```

Figura 27 – Comportamiento de recepción de información

Movimiento del robot

Por último, se le asigna al agente robot un comportamiento que se ejecutará una única vez y que se llamará "comportamiento de navegación", donde deberá mover al robot acorde a las posiciones deseadas y de los datos recibidos por los agentes dispositivos. En la figura 28 se muestra el algoritmo seguido por este comportamiento donde, mientras exista una nueva posición a alcanzar, el robot se desplazará hacia la posición deseada. Para conocer la posición en la que se encuentra realizará un cálculo en la que intervienen todos los agentes dispositivos incluidos. A continuación, se explica cómo se lleva a cabo este cálculo.

```
Algoritmo MovimientoRobot
        Mientras Existe ( NuevaPosicion ) Hacer
                Leer NuevaPosicion
                Mientras Posicion != NuevaPosicion Hacer
                        MoverA( NuevaPosicion )
                        Posicion = CalcularPosicion()
                Fin Mientras
        Fin Mientras
FinAlgoritmo
```

Figura 28 – Comportamiento del movimiento del robot

Cálculo de la posición del robot

El agente robot, puede recibir una gran información sobre su posición de diversos dispositivos de sensorización y tiene que tratar esta información de manera adecuada. En el comportamiento de recepción de información, se había almacenado la información sobre la posición del robot junto al nombre del agente que la ha mandado y la hora de la recepción. Gracias a estos datos, se puede implementar un modelo de confianza basado en los datos recibidos y en el tiempo que hace que los ha recibido. Esta confianza representa la creencia del robot hacia la información recibida de los dispositivos de sensorización. Cuanto más confíe en la posición ofrecida por un dispositivo, mayor confianza tendrá, y al revés para el caso contrario. Además, se utilizará esta confianza asignada a los dispositivos de sensorización para calcular la posición actual del robot. Los pasos que seguirá el agente robot para calcular la posición en la que se encuentra y para ajustar la confianza de los dispositivos de sensorización serán los siguientes:

1) Cuando el robot termina un movimiento comprueba si la posición en la que se encuentra es la que realmente quería alcanzar. Para poder realizar esto, le asigna una probabilidad a cada posición devuelta por los dispositivos de sensorización dependiendo del tiempo (α) que hace que mandó la información (a mayor diferencia, más próximo a cero) y de la confianza (β) que tenga sobre él (inicialmente 1). En la figura 29, se puede observar cómo se calcula la posición del robot (compuesto por los valores x e y). $\hat{p}_{\{x,y\}}$ es el valor de la posición final que creerá que tiene el robot, mientras que $p_{i\,\{x,y\}}$ es la posición que cree que tiene un dispositivo de sensorización i de los d existentes.

$$\hat{p}_{\{x,y\}} = \frac{\sum_{i=0}^{d} \alpha_i \cdot \beta_i \cdot p_{i\,\{x,y\}}}{\sum_{i=0}^{d} \alpha_i \cdot \beta_i}$$

Figura 29 – Cálculo de la posición del robot

2) Después de calcular la posición del robot, se realiza una modificación de la confianza de los dispositivos de sensorización dependiendo de la distancia entre la posición obtenida y la posición calculada en el paso 1, y de esta distancia en el movimiento anterior. En la figura 30 aparece la tabla que se emplea para modificar esta confianza. Debido a que en este proyecto se ha realizado una división por casillas del escenario, el error obtenido es una medida a base de casillas. De esta forma, si en la posición dada por un dispositivo de sensorización coincide con la posición calculada, se aumentará la confianza sobre el dispositivo de sensorización sin sobrepasar el valor de 1. Por otro lado, si no coincide con la calculada existen dos posibilidades. La primera, consiste en que la distancia actual sea de una única casilla y que la posición anterior fuera correcta, con lo que no se aumentará ni disminuirá el valor de la confianza. También, existe la posibilidad de que la distancia actual sea de una única casilla y que la posición anterior no fuera la correcta o de que la distancia actual fuera de más de una casilla, con lo que se disminuirá el valor de la confianza sin sobrepasar el valor de 0. El robot aquí

también juega un papel importante, ya que él también opina sobre en qué posición se encuentra y también tiene un valor de confianza sobre su creencia.

Distancia movimiento previo	Distancia movimiento actual	Modificación confianza
0	0	+ 0.10
1	0	+ 0.05
0	1	0.00
>0	1	- 0.05
-	>1	- 0.10

Figura 30 – Tabla de valores para calcular la confianza de los dispositivos de sensorización

3) Si el valor obtenido al calcular la posición con la ayuda de los dispositivos de sensorización es el mismo al que el robot deseaba ir, se mueve a la siguiente posición. En el caso de que no coincidan las posiciones, vuelve a moverse desde la nueva posición en la que se encuentra, la previamente calculada, hacia la que desea moverse. Estos pasos se vuelven a repetir hasta que se logra llegar a realizar el recorrido deseado, consiguiendo alcanzar cada posición de este.

5.2. Agente dispositivo

En el agente dispositivo se han diseñado varios comportamientos, distintos a los utilizados por el agente robot. La figura 31 se corresponde al conjunto de comportamientos seguido por el agente dispositivo. En primer lugar, se dispone del registro del agente en el sistema multiagente a través del AMS, como en el caso del agente robot. Por tanto, también deberá seguir los pasos propuestos por la plataforma *SPADE*, asignándole un identificador llamado *Jabber ID* (JID), seguido del *host* en el que se esté ejecutando el sistema multiagente, y una contraseña con la que conectarse. A continuación, se detallará la manera de actuar del agente dispositivo.

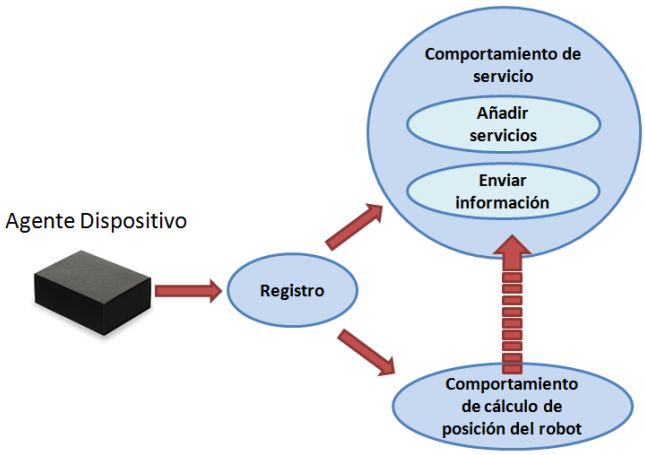


Figura 31 – Comportamientos seguidos por el agente dispositivo

Cálculo de la posición del robot

Después de la inicialización del agente dispositivo en el sistema multiagente, el agente deberá realizar un cálculo de la posición en la que se encuentra el robot. Para realizar este cálculo, se añade al agente dispositivo un comportamiento cíclico donde calculará constantemente la posición del robot, que se llamará "comportamiento de cálculo de posición del robot". Este comportamiento es diferente en cada dispositivo de sensorización, ya que el cálculo de la posición varía dependiendo de dónde se encuentre éste y del sensor que tenga. Además, cada dispositivo de sensorización puede tener más de un sensor, por lo que el agente puede tener replicado este comportamiento, ofreciendo la posición calculada por cada sensor por separado. Como este proyecto se realiza utilizando cámaras como dispositivos de sensorización, se utiliza la librería de *OpenCV* en este comportamiento para desarrollar un algoritmo que reconozca la posición del robot. Como ya se ha comentado en la sección 4, el escenario utilizado estaba compuesto por 36 cuadrículas, por eso, la manera de calcular en qué posición se encontraba el robot era la siguiente:

- Primero, se realiza una búsqueda del robot en el eje horizontal (eje x) y se almacena la posición en la que mayor espacio ocupe.
- Después, se procede a otra búsqueda del robot, pero, esta vez, en el eje vertical (eje y). Esta búsqueda es distinta a la anterior, ya que, en este caso, se busca al robot desde la posición más próxima a la cámara hasta la más lejana y se almacena la posición en la que aparece el robot en primer lugar siempre que el área que ocupe supere cierto umbral, ya que el robot pertenecerá a la casilla que ocupe mayor área y en ocasiones puede ocupar parte de una casilla y no por eso pertenecer a esta. El umbral se escogió a través de diversas pruebas realizadas con el robot. Utilizando el dispositivo *Xtion PRO LIVE*, introducido en el apartado 4, resulta más sencillo realizar este paso, ya que se realizaría un barrido de profundidad y fácilmente podríamos calcular donde se encuentra.

En la figura 32 se puede apreciar la realización de esta búsqueda de posición por un agente dispositivo. En el eje *x*, encontraría la posición 6, que corresponde a la selección del centro, mientras que en el eje *y*, encontraría la posición 2, que también corresponde al centro. Se puede observar que, si en la búsqueda en el eje vertical se hubiera seleccionado al de mayor probabilidad, se hubiera escogido la posición 3, que corresponde a la más alejada de la cámara y es una posición errónea.

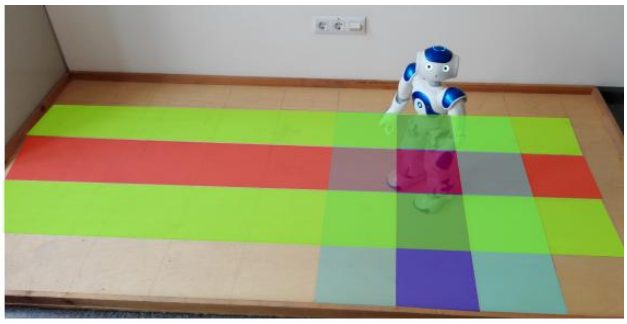


Figura 32 – Búsqueda de la posición del robot

Añadir servicio

Otra de las características de estos agentes es la utilización de servicios donde se deberá añadir cada uno de los servicios que ofrece al DF. Este servicio consiste en informar sobre la posición del robot. Cabe destacar que, si el dispositivo tuviera más de un sensor con el que pueda conocer la posición del robot, se crearían tantos servicios como sensores tuviera, al igual que se crearían también el mismo número de comportamientos de cálculo de posición del robot. Estos servicios se ejecutan dentro de un comportamiento cíclico, llamados "comportamiento de servicios", donde observa qué sensores tiene disponibles y activa o desactiva estos servicios según su disponibilidad.

La manera en la que se añade el servicio al DF en *SPADE* se puede observar en la figura 33, donde primero se crea la descripción de un agente, al que le asignamos un servicio con el nombre que deseemos. A este servicio se le puede añadir una gran variedad de información, pero en el ejemplo mostrado únicamente se busca indicar la disponibilidad del servicio. Por último, se registra el servicio y después se crea un evento indicando el nombre de la publicación de la posición. Resumiendo, se crea un servicio que indica la disponibilidad del evento, al que el agente robot debe subscribirse.

```
dad     = spade.DF.DfAgentDescription()
service = spade.DF.ServiceDescription()
service.setName("Camera")
service.setType("Available")
addService(service)
registerService(dad)
createEvent( EventName )
```

Figura 33 – Creación de un servicio en *SPADE*

Enviar información

Por último, como se ha explicado en la sección 5.1 se va a hacer uso del protocolo de publicación y subscripción de *SPADE*, el cual permite a un agente crear un evento o subscribirse a los eventos creados por otros agentes a los que se está subscrito a través de la notificación de presencia. Dicho esto, a través del evento creado en la figura 33, se realizará la publicación de la información sobre la posición del robot obtenida, gracias a los sensores del dispositivo, a cada uno de los agentes subscritos al servicio. Como se está utilizando el mecanismo de publicación y subscripción, únicamente se necesitará una instrucción para realizar el envío del mensaje a todos los agentes subscritos que se puede apreciar en la figura 34.

```
publishEvent( EventName, RobotPosition )
```

Figura 34 – Publicación de eventos

Tal y como se ha explicado en las secciones 4.1 y 4.2, los comportamientos de los dos agentes son muy distintos, pero colaboran en el sistema multiagente para ayudar al robot a conocer su posición. En la figura 35 se puede visualizar el comportamiento de estos agentes según su interacción con la plataforma *SPADE*, donde ambos se registran inicialmente en el AMS. Después, el agente dispositivo añade los servicios que ofrece al DF y crea el evento que utiliza el protocolo de publicación y subscripción, a diferencia del agente robot, que busca los servicios disponibles a través del DF y se subscribe a los eventos de estos servicios. Por último, el agente dispositivo publica la información de la posición del robot, obtenida a través de sus sensores, mientras que el agente robot recibe esta información, gracias a la subscripción realizada, y la utiliza para conocer la posición en la que se encuentra.

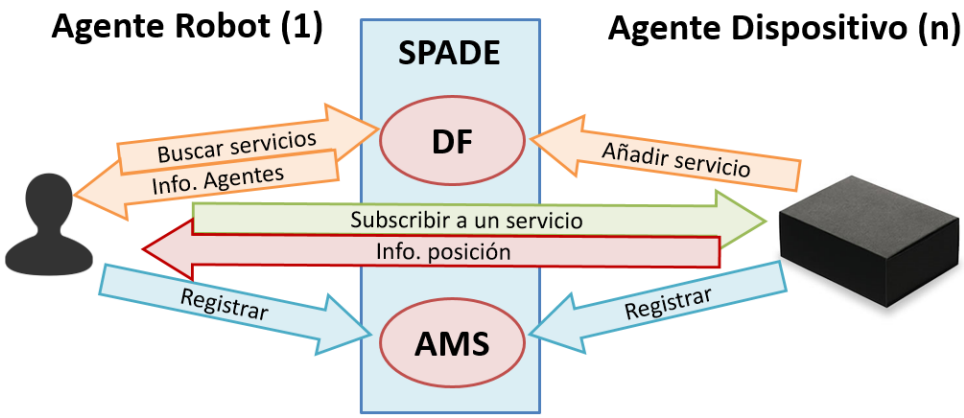


Figura 35 –Visión general del sistema propuesto

Una vez planteado el diseño del sistema multiagente, el siguiente paso es el desarrollo del procedimiento experimental, el cual se divide en tres pruebas. La primera, consistirá en realizar pruebas donde el agente robot está introducido dentro de un sistema multiagente que dispone de agentes dispositivos, los cuales conocen de forma muy precisa la posición del robot. En estas pruebas, se estudiará el error entre la posición final obtenida y la posición final deseada, aumentando el número de dispositivos de sensorización. En la segunda prueba, se va a llevar a cabo la utilización de agentes dispositivos que tienen acumulado cierto error. Se realizará el mismo estudio que en la prueba anterior y además se estudiará como varía la confianza del agente robot hacia los dispositivos de sensorización según su error acumulado. Finalmente, en la última, se observará cómo funciona el sistema desarrollado en un escenario dividido por zonas, ya que es el objetivo principal que se perseguía en este trabajo.

6. Pruebas y evaluación

Después de haber seleccionado el pavimento en el que el robot va a desplazarse y de haber desarrollado los recorridos que va a realizar sobre éste, es hora de evaluar el sistema multiagente desarrollado. Para ello, se han utilizado cámaras como dispositivos de sensorización en estas pruebas.

Para medir el error cometido por las pruebas, se va a utilizar la distancia euclídea, donde se trata de medir la distancia mínima entre las dos casillas (la que se encuentra el robot respecto a la deseada), es decir, la longitud de la recta que une la posición en la que se encuentra el robot con la deseada. Por último, destacar que, para obtener unos resultados estadísticamente fiables, se ha realizado un total de 10 repeticiones de cada una de las pruebas, por lo que se mostrará en gráficas tanto la media del error obtenido en cada movimiento realizado como el error medio estándar.

6.1. Pruebas con cámaras sin error

En esta sección, se van a realizar diferentes pruebas sobre el recorrido circular, el recorrido estrella pitagórica y el recorrido zigzag. En ellas, se busca encontrar el número mínimo de cámaras utilizables para corregir el error, hasta que éste desaparezca. Por eso, se utilizaron diferentes cámaras que conocían la posición exacta del robot y se probó a realizar los diferentes recorridos aumentando de uno en uno el número de cámaras cada vez, empezando inicialmente sin cámaras.

Las primeras pruebas que se realizaron fueron del recorrido circular. En la figura 36 se puede observar una gráfica donde aparece el error obtenido, además del error medio estándar, a medida que se van realizando movimientos a diferentes posiciones. Como se ha comentado al inicio de la sección, en la gráfica se encuentran los resultados utilizando diferente número de cámaras. Sin utilizar ninguna cámara, el error cometido crece de manera rápida en los primeros tres movimientos, pero se mantiene oscilando en el resto hasta obtener un error de *0.80* casillas (unidad de medida utilizada, ya que el escenario está dividido por casillas). Por otro lado, utilizando una cámara se consigue reducir este error. El robot no siempre alcanza la posición deseada, lo cual se debe a que, además de la posición ofrecida por la cámara, se tiene la posición en la que el robot cree que se encuentra, y llega a una posición intermedia entre ambas. Aun así, utilizando una única cámara, se consigue reducir el error cometido por el robot de *0.80* casillas a *0.50* casillas, es decir, se ha reducido un *37.5%* el error. Después, se decidió realizar una prueba utilizando dos cámaras y se obtuvieron los mejores resultados posibles, consiguiendo que el robot alcanzase la posición deseada en cada uno de sus movimientos, es decir, se consiguió reducir el error en un *100%*.

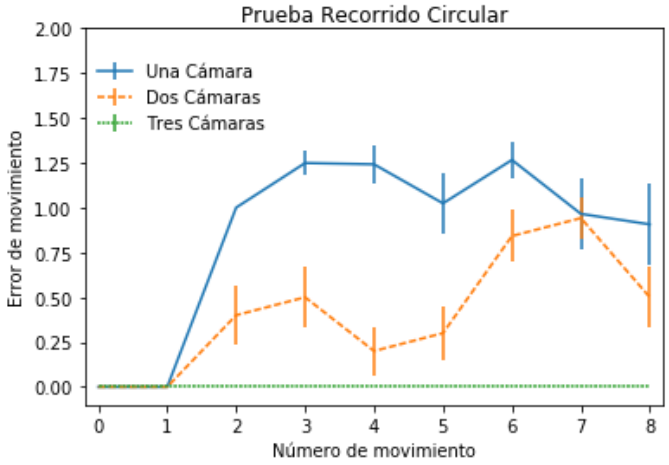


Figura 36 – Resultados del recorrido circular con cámaras que conocen la posición exacta del robot

Tras las pruebas de recorrido circular, se realizaron las pruebas sobre el recorrido estrella pitagórica. En la figura 37 se muestra una gráfica con los resultados obtenidos por estas pruebas. Primero, se realizó una prueba sin utilizar ninguna cámara, y se pudo comprobar que el error obtenido en cada uno de los movimientos se mantenía siempre cercano a *1* casilla, disminuyendo los valores en comparación con el recorrido circular. Esta disminución del error se debe a que el anterior circuito es un recorrido más largo, poseyendo un mayor área de trabajo, 4*x*4, en comparación al área de trabajo utilizado por este recorrido, 3*x*3. Al añadir una cámara, el error obtenido disminuía, reduciendo el error cometido por el robot de *0.60* casillas a *0.50* casillas, que equivale a una mejora del *16.67%*. Esta mejora es menor que la obtenida por el recorrido circular de *37.5%*, debido al área de trabajo utilizado. Por último, se realizó una prueba con dos cámaras. Como en el recorrido circular, en la prueba se consiguió que el robot alcanzase la posición deseada en cada uno de sus movimientos, reduciendo su error en un *100%*.

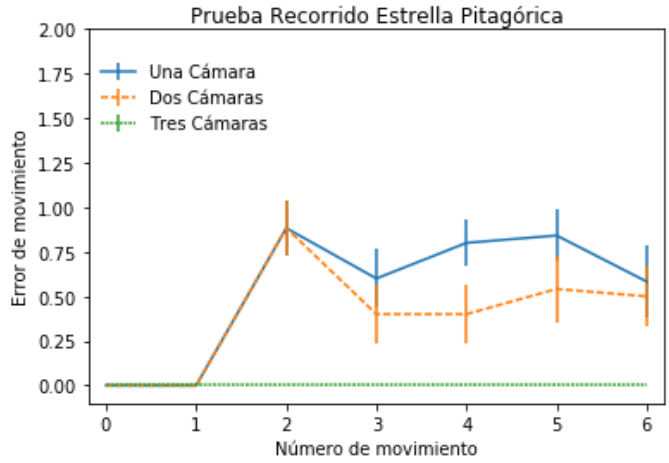


Figura 37 – Resultados del recorrido estrella pitagórica con cámaras que conocen la posición exacta del robot

Por último, se realizaron las pruebas del recorrido zigzag. La figura 38 hace referencia a los resultados obtenidos al realizar las pruebas sobre este recorrido. En la prueba sin

cámaras, se puede observar que, a diferencia de los anteriores recorridos, el error de los movimientos oscila entre *1* y *2* casillas. Este resultado se debe a que, en ciertas ocasiones, el robot se corrige solo debido a la aleatoriedad de su movimiento. El resultado final obtenido por este recorrido es de *1.40* casillas, el mayor error obtenido hasta ahora. Esta diferencia es consecuencia de que el recorrido es mucho más largo que los anteriores, aumentando el error acumulado. Después, se llevó a cabo una prueba utilizando únicamente una cámara. En ésta, también, se puede observar que los valores oscilan entre *0.50* y *1.50* casillas, ya que, además de la posición ofrecida por la cámara, se dispone de la posición en la que el robot cree que se encuentra, y llega a una posición intermedia entre ambas. El error final obtenido utilizando únicamente una cámara es de *1.2* casillas, se consiguió una mejora del *14,29%*. Después, como en los anteriores recorridos, se realizó una prueba con dos cámaras. En esta prueba, se obtuvieron los mejores resultados posibles, alcanzando siempre la posición deseada en cada movimiento del robot, al igual que en los anteriores recorridos, es decir, se redujo su error en un *100%*.

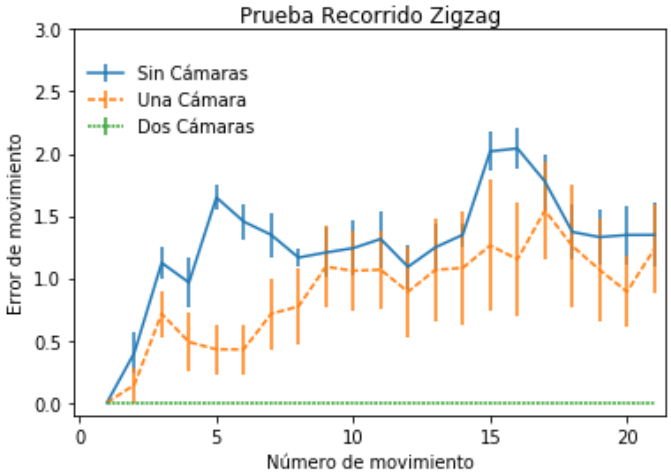


Figura 38 – Resultados del recorrido zigzag con cámaras que conocen la posición exacta del robot

En conclusión, se puede decir que, utilizando el sistema multiagente diseñado en este trabajo, se mejoran los resultados obtenidos por el robot al desplazarse. En estas pruebas, se han utilizado cámaras que conocen de manera muy precisa la posición del robot, y se ha podido comprobar que, al introducir únicamente dos cámaras con estas características se puede mejorar los resultados hasta el punto de que el robot se mueva siempre a la posición deseada. Por otro lado, si se utilizara únicamente una cámara, no se podría asegurar que el error se corrigiera del todo, pero sí que mejorase.

Estas pruebas realizadas utilizan cámaras ideales, donde conocen la posición exacta del robot y donde la información sobre la posición siempre es enviada. A continuación, se van a explicar diferentes pruebas donde se utilizan cámaras que contienen cierto error en su información sobre la posición del robot.

6.2. Pruebas usando cámaras con error

Después de realizar las pruebas con cámaras sin error, se comprobó que si se utilizaran dos cámaras sin error se podrían obtener los mejores resultados posibles. En un escenario ideal sucedería esto, pero en un escenario real pueden existir diferentes motivos por el cual un dispositivo de sensorización deje de funcionar correctamente, ya sea porque sus sensores han perdido eficacia o porque haya cambios en el entorno que puedan entorpecer su identificación de posición, entre otras causas. Por ese motivo, se decidió realizar diferentes pruebas con cámaras que contienen cierto error en su información sobre la posición del robot, intentando simular lo que ocurriría en un entorno real.

La primera prueba que se realizó fue la de calcular el error entre la posición final obtenida y la posición final deseada, aumentando el número de cámaras. Para ello, se seleccionó el recorrido estrella pitagórica, ya que era en que mejores resultados se obtenían. Las cámaras utilizadas en estas pruebas tenían un error aleatorio entre *10%* y *30%*. En la figura 39 aparece una gráfica donde se muestra el error obtenido por las pruebas aumentando el número de cámaras, además del error medio estándar de cada una, a medida que se va realizando movimientos a diferentes posiciones. En la gráfica se puede ver que, cuando únicamente existe una cámara en el sistema multiagente, el error aumenta paulatinamente. El error final utilizando una cámara es de 1.1 casillas, aumentando el error obtenido al no utilizar ninguna cámara, *0.6* casillas, que se puede observar en la figura 37. A medida que aumenta el número de cámaras, el error que se obtiene se va estabilizando, consiguiendo unos valores del error final de *0.4*, *0.2* y *0* casillas empleando 2, 3 y 4 cámaras, respectivamente.

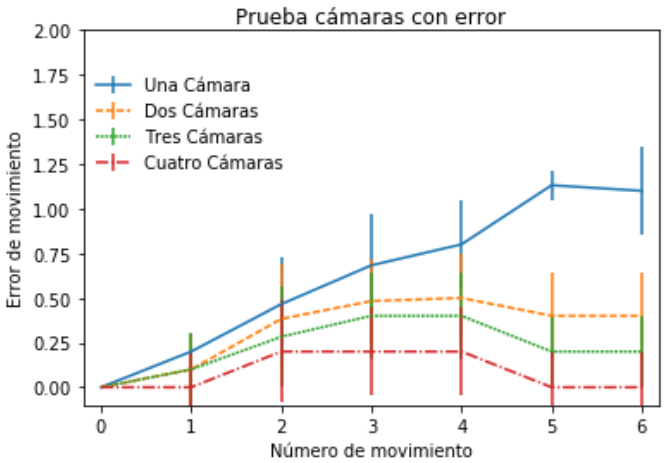
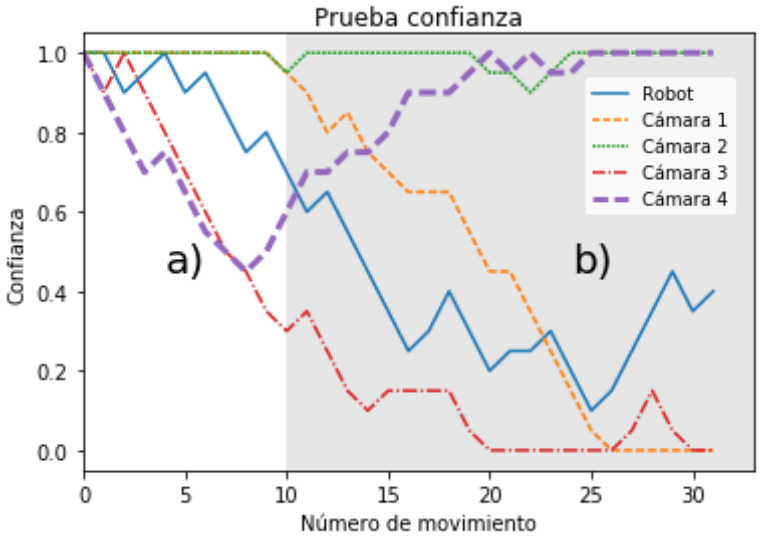


Figura 39 – Resultados del recorrido estrella pitagórica con cámaras que tienen un error entre el *10%* y el *30%* sobre la posición del robot

Utilizando 4 cámaras, se ha logrado alcanzar la posición final sin ningún error en ninguna de las 10 pruebas realizadas, pero, debido a que las cámaras no son precisas, en algunas de estas pruebas no se llega a alcanzar la posición deseada en alguno de los pasos intermedios. Como el error obtenido en dichos pasos intermedios era de entre *0* y

0.2 casillas, se consideró que se trataba de un error bastante aceptable y no se realizaron más pruebas aumentando el número de cámaras.

Como se comentó en la sección 5.1, el agente robot asigna confianza a los diferentes dispositivos de sensorización según la tabla vista en la figura 30. Esta confianza le sirve al agente robot para poder calcular la posición en la que se encuentra el robot, añadiéndole más peso a la información ofrecida por aquellos dispositivos que poseen mayor confianza. Por este motivo, se decidió realizar una prueba para observar cómo variaba la confianza en un conjunto de cámaras y qué sucedía si alguna de las cámaras actuaba de manera diferente en un momento dado, es decir, se buscaba observar el comportamiento del sistema multiagente diseñado y ver si la confianza ofrecida por el robot a los diferentes dispositivos de sensorización era la adecuada. Para ello, se llevó a cabo una prueba con cuatro cámaras y el robot, ya que también se le asigna una confianza a la posición en la que el robot cree que se encuentra. La figura 40 hace referencia a la gráfica donde se muestra la confianza dada a los diferentes dispositivos de sensorización. Inicialmente, las cámaras 1 y 2 tenían un error del *10%*, mientras que las cámaras 3 y 4 tenían un *20%* y *40%*, respectivamente. Todas las confianzas empiezan en 1 y, seguidamente, tanto la creencia del robot como las cámaras 3 y 4 van disminuyendo su confianza, ya que son las que mayor error acumulan. Después, a partir del movimiento número 10, las cámaras 1 y 4 sufren un intercambio de error obteniendo un error del *40%* y *10%,* respectivamente, y se puede ver como la cámara 1 va disminuyendo su confianza hasta alcanzar el valor de 0, mientras que la cámara 4 aumenta su confianza hasta el valor 1, que es lo máximo posible. Se puede observar, el agente robot diseñado les asigna una mayor confianza a aquellos dispositivos de sensorización que menor porcentaje de error llevan acumulado, y que funciona correctamente ante los posibles cambios generados por los diferentes agentes dispositivos.



Porcentaje de error a)
Cámara 1 -> 10 %
Cámara 2 -> 10 %
Cámara 3 -> 20 %
Cámara 4 -> 40 %

Porcentaje de error b)
Cámara 1 -> 40 %
Cámara 2 -> 10 %
Cámara 3 -> 20 %
Cámara 4 -> 10 %

Figura 40 – Estudio de la confianza del robot hacia las cámaras

6.3. Pruebas en un escenario distribuido

La finalidad de este proyecto era intentar ayudar al robot NAO a alcanzar la posición deseada a través de cámaras fijas en un entorno compuesto por diferentes zonas. Pero hasta ahora, las pruebas realizadas han servido para comprobar el correcto funcionamiento del sistema multiagente en una única sala. Una vez comprobado lo anterior, es hora de realizar pruebas por un escenario compuesto por diferentes salas, donde cada una de ellas dispone de un número distinto de cámaras que sólo informan cuando ven al robot. En las pruebas, se ha realizado un recorrido de 7 movimientos sobre dos salas o zonas, como se puede apreciar en la figura 41. Como el entorno disponible era tan reducido, únicamente se han utilizado dos salas, ya que si no los resultados no serían apreciables.

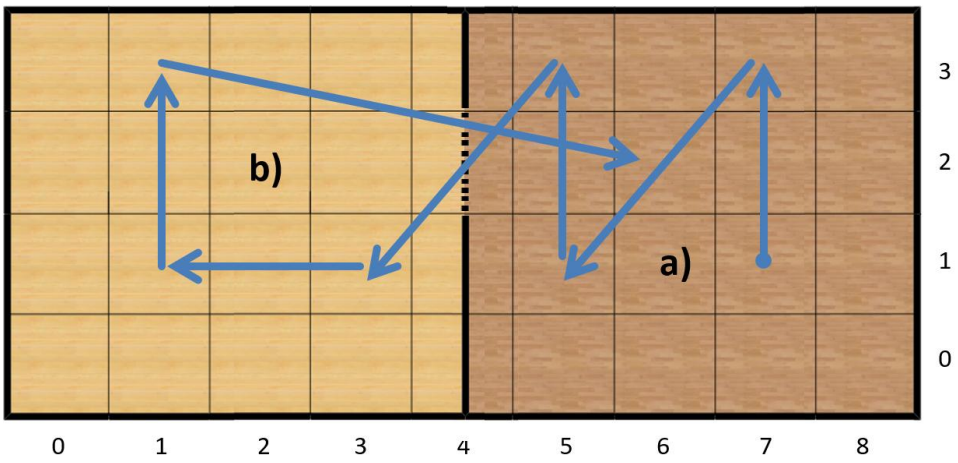


Figura 41 – Recorrido en un entorno compuesto por dos salas

Por un lado, tomando como referencia los resultados obtenidos en la figura 39 de la sección 6.2, se han utilizado cuatro cámaras con un porcentaje de error del *10%* al *30%* para la zona *a)*, ya que con ese número de cámaras se obtienen unos valores aceptables. También, se podrían haber utilizado dos cámaras ideales, pero en estas pruebas se intenta simular un entorno real donde es más probable que las cámaras tengan un cierto error acumulado. Por otro lado, en la zona *b)* se han utilizado dos cámaras con un error entre el *50%* y el *70%*. Lo que se intenta demostrar en esta prueba es cómo el robot consigue alcanzar, en la mayoría de los casos, la posición deseada de manera correcta en la zona *a),* y que al introducirse en la zona *b)* comienza a acumularse el error. Además, una vez que el robot vuelve a la zona *a),* se consigue reducir este error acumulado de la zona *b)*. En la figura 42 se muestra el error obtenido, además del error medio estándar, ya que se realizan 10 pruebas distintas, a medida que se va desplazando a diferentes posiciones. En la gráfica, se puede distinguir el error obtenido en las dos zonas, donde los errores de la zona *a)* están representados con un fondo blanco, mientras que los errores de la zona *b)* aparecen con un fondo gris. Se puede concluir que los resultados son como los que se esperaban obtener, donde en la zona *a)* se consigue alcanzar, en la mayor parte de los casos, la posición correcta, obteniendo

como máximo un error de *0.2* casillas. Además, como se esperaba, al introducirse en la zona *b)*, donde se dispone de unas cámaras más imprecisas, el error aumenta hasta alcanzar *1* casilla, que supone un error 5 veces mayor respecto a la zona anterior. Finalmente, al volver a la zona *a)*, se consigue reducir el error acumulado en la zona *b)*, obteniendo un error de sólo *0.2* casillas.

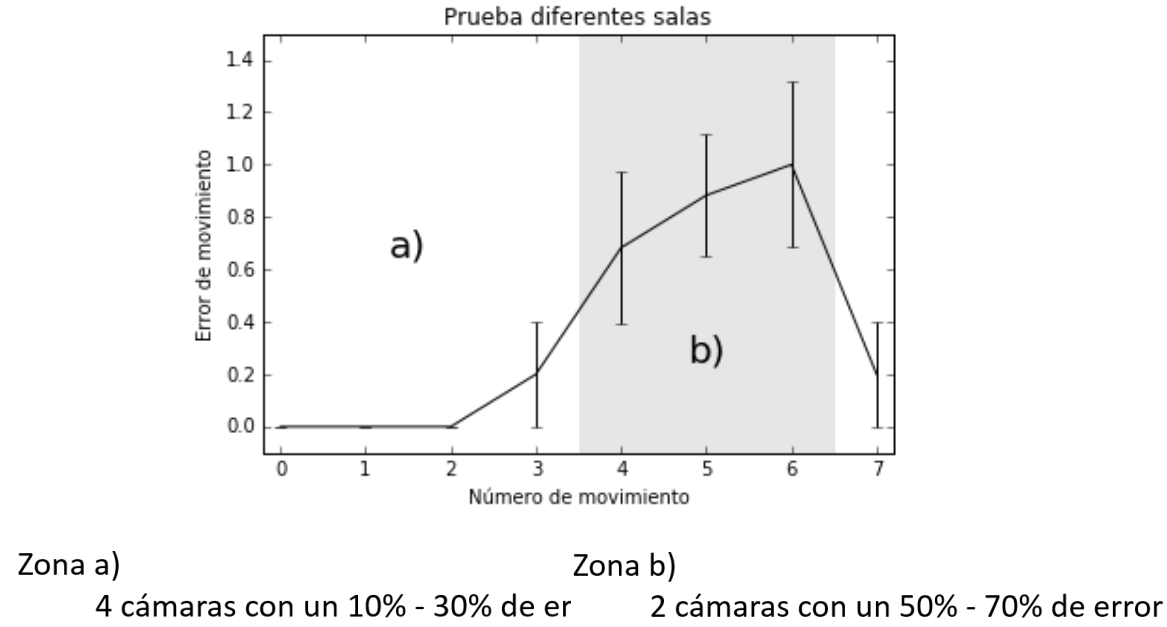


Figura 42 – Resultados obtenidos en un entorno compuesto por dos salas

Recapitulando sobre las pruebas realizadas, en la primera prueba se buscaba introducir al robot en un escenario ideal, donde los dispositivos de sensorización, concretamente cámaras, informan siempre sobre la posición real del robot, sin ninguna pérdida de información. En esta prueba, se observó que con dos cámaras ideales era suficiente para ayudar al robot a conocer su posición en cada instante. Después, se realizó esta misma prueba utilizando dispositivos de sensorización que poseían cierto umbral de error en su información, y se observó que, utilizando cuatro de estos dispositivos, se podía obtener la posición deseada, pudiendo cometer un error de *0.2* casillas. También, se analizó un estudio de la confianza de los diferentes dispositivos de sensorización y se determinó que se le asignaba mayor confianza a aquellos que poseían menor error, que era precisamente lo que se buscaba conseguir. Por último, en esta prueba se ha podido observar que se han cumplido los objetivos propuestos, ya que se ha desarrollado un sistema multiagente que, gracias a la utilización de la notificación de presencia de *SPADE*, permite la incorporación y el abandono de diferentes dispositivos de sensorización, según su disponibilidad, de manera transparente. Dicho esto, a continuación, se detallará las conclusiones y las líneas futuras de actuación.

7. Conclusiones

La necesidad de la localización del robot surgió a la hora de realizar diferentes proyectos donde éste se debía desplazar por un entorno. En ellos, se pudo observar que el movimiento del robot no era regular y, por este motivo, se decidió realizar una localización diferente a las existentes, donde el robot interactúa en un escenario compuesto por distintas salas o zonas. La novedad de este escenario es que cada zona está compuesta por dispositivos de sensorización propios, diferentes a los dispositivos de otras zonas, que informan de la posición en la que se encuentra el robot. El objetivo que este trabajo perseguía era ayudar a alcanzar la posición deseada utilizando una solución flexible y distribuida mediante un sistema multiagente. Además, el sistema multiagente permite que diferentes dispositivos de sensorización se incorporen o salgan de forma dinámica y transparente. Por otro lado, la solución propuesta también permitirá la incorporación de cualquier tipo de dispositivo de sensorización que puede dar información sobre la posición del robot.

En este capítulo se presentarán los objetivos cumplidos en este trabajo, además del posible trabajo futuro que se puede realizar.

7.1. Objetivos cumplidos

De acuerdo con los objetivos indicados inicialmente en este trabajo, se ha realizado un estudio sobre las diferentes herramientas y características de los agentes inteligentes y sistemas multiagente. Gracias a este estudio, se seleccionó la herramienta *SPADE* para diseñar el sistema multiagente y modelar los distintos agentes utilizados en el trabajo.

Además del estudio anterior, se llevó a cabo otro estudio sobre las características del robot NAO y sus especificaciones, ya que era el robot que se iba a utilizar en el proyecto. Con este estudio, se obtuvieron conocimientos teóricos sobre la programación del robot y de las posibles herramientas que se podían utilizar.

Después, se procedió al diseño del escenario donde se observó que el robot se desplazaba mejor sobre la madera, por lo que se utilizó un escenario de madera que se dividió en 36 partes (9 verticales por 4 horizontales) de un tamaño de $25x25\ cm^2$. Por este motivo, la posición del robot, en este proyecto, está formada por los valores del eje x e y, y por tanto, cuando se habla del error cometido por el robot se emplea la unidad de medida de casillas, ya que tratará del número de casillas de distancia que se encuentre el robot de la posición deseada.

El siguiente paso que se realizó fue desarrollar los diferentes agentes que componen el sistema multiagente. Concretamente, se crearon dos agentes diferentes: el agente robot, que se encargaba de controlar al robot, y el agente dispositivo, que controlaba al dispositivo que obtiene la posición del robot a través de sensores.

Por un lado, el agente robot tenía tres comportamientos diferentes: el comportamiento de búsqueda de servicios, que buscaba todos los servicios de manera cíclica y se suscribía a los disponibles, el comportamiento de recepción de información, que almacenaba todos los mensajes devueltos por la suscripción anterior, y el comportamiento de navegación que, como su propio nombre indica, se encargaba de realizar el movimiento del robot. Por otro lado, el agente dispositivo poseía el comportamiento de cálculo de posición del robot, que era un comportamiento cíclico que tenía la función de almacenar la posición del robot, y el comportamiento de servicio, cuya finalidad era de dar de alta los diferentes servicios que ofrece y de enviar la posición obtenida por el comportamiento anterior. Este agente se replicó en las pruebas para obtener distintos agentes con los que formar el sistema multiagente. Para ello, se realizó una modificación en el comportamiento de cálculo de posición del robot, ya que dependía de la posición en la que se encontraba el dispositivo de sensorización. Además, en el diseño de los agentes se utilizó la notificación de presencia propia de la herramienta *SPADE*. Gracias a esta característica, se podía conocer en cada momento qué cámaras estaban disponibles y cuáles no.

Por último, se diseñaron diferentes pruebas con el fin de comprobar el funcionamiento del sistema multiagente diseñado. Inicialmente, se realizaron unas pruebas utilizando cámaras ideales, las cuales conocían de forma precisa la posición real del robot. En ellas, se aumentaban el número de cámaras, inicialmente sin cámaras hasta un total de dos, para conocer cuántas hacen falta para corregir el error cometido en el desplazamiento del robot por completo. Para realizar estas pruebas se utilizaron tres recorridos diferentes donde en cada uno de ellos se realizaron treinta pruebas distintas, concretamente diez pruebas por cada nuevo sistema, es decir, diez pruebas sin cámaras, otras diez con una cámara y así sucesivamente. Realizando diez veces cada prueba por sistema, se buscaba conseguir unos resultados estadísticamente fiables. Estos resultados obtenidos son muy prometedores, porque, inicialmente, sin utilizar ninguna cámara en el sistema multiagente, el robot obtenía un error entre *0.60 y 1.40* casillas, que se mejoraron al añadir una cámara al sistema, consiguiendo un error de entre *0.5 y 1.2* casillas. Se observó que añadiendo únicamente una cámara, se conseguía reducir el error cometido por el robot, pero no disminuía lo suficiente, lo cual se debe a que, además de la posición correcta que proporciona la cámara, también se toma como referencia la creencia del robot sobre su posición. Al aumentar el número de cámaras en dos, se mejoraron los resultados obtenidos por el robot y se consiguió que el error cometido se corrigiera a la perfección.

Las pruebas anteriores simulaban un escenario ideal, donde todas las cámaras conocían la posición real del robot y se demostró que, utilizando únicamente dos cámaras ideales, el robot podía alcanzar su posición deseada en cada uno de sus movimientos. La siguiente prueba que se llevó a cabo consistía, básicamente, en lo mismo que la prueba anterior: aumentar el número de cámaras hasta ver cuántas hacían falta para corregir el error cometido por el robot, pero, esta vez, utilizando cámaras que llevaran incorporadas cierto error en su información. Esta prueba intentaba simular un escenario real, donde pueden existir diversos factores ambientales o diferentes objetos en medio del escenario que dificulten el conocimiento de la posición real del robot. Se utilizó hasta un máximo de cuatro cámaras con un error que estaba comprendido entre el *10%* y el *30%*, y como en la prueba anterior, se realizaban diez pruebas por sistema, realizando un total de 40 pruebas. Utilizando una única cámara en el sistema, se

obtenía un error final de 1.1 casillas, el cual consiste en la distancia en la que se encuentra el robot tras realizar su último movimiento respecto a la posición final deseada. Aumentando el número de cámaras, se consiguió disminuir este error hasta corregirlo por completo al utilizar cuatro cámaras. Cabe destacar que en algunos de los movimientos intermedios se obtiene un error de *0.2* casillas, pero que al deberse a un error tan reducido, se ha considerado un error aceptable.

Otra de las pruebas realizadas, ligada a la prueba anterior, consistía en observar como variaba la confianza que el robot tiene sobre los dispositivos de sensorización que poseen cierto error, y cómo varía ésta al cambiar el error de los dispositivos de sensorización. Tras la prueba realizada en la sección 6.2, se observó que actuaba de manera correcta, asignando mayor confianza a aquellos dispositivos de sensorización que menor porcentaje de error poseían. Además, al hacer el cambio de error entre dos de los dispositivos de sensorización, se observó que su confianza se ajustaba asignando mayor confianza al dispositivo que disminuyó su error y menor confianza al dispositivo que aumento su error, tal y como se esperaba.

Como el objetivo de este trabajo era ayudar al robot a conocer su posición en un escenario compuesto por diferentes zonas donde cada zona posee unas cámaras propias, se realizó una última prueba en la que el robot debía desplazarse por dos zonas. La primera de las zonas, que era donde se encontraba el robot, estaba compuesta por cuatro cámaras con un error entre el *10%* y el *30%*, ya que en pruebas anteriores se observó que se podían obtener buenos resultados utilizando estas características. La segunda zona poseía únicamente dos cámaras con un error entre el *50%* y el *70%*. En esta prueba, se pretendía demostrar cómo el robot se desplaza correctamente por la primera zona, pero que al entrar en la segunda su error aumenta considerablemente. Por último, el robot debía volver a la zona inicial en la que seguiría alcanzando la posición deseada, ya que la confianza en los dispositivos de sensorización de esa zona no se ha visto afectada al cambio de zona. Los resultados obtenidos eran tal y como esperábamos, donde en la zona inicial se obtenía un error entre *0* y *0.2* casillas, mientras que en la segunda zona, se conseguía un error de hasta *1* casilla, 5 veces mayor que la zona anterior.

A modo de resumen, se ha desarrollado un sistema multiagente donde se pueden incorporar diferentes dispositivos de sensorización que ayudan a la localización del robot en un entorno compuesto por múltiples salas, gracias a diferentes sensores que poseen. Además, se ha demostrado que, aunque el conjunto de información obtenida tenga un porcentaje de error, cuanta más riqueza de información se posea, mejores resultados se obtendrán. En este proyecto, se han utilizado cámaras como sensores, pero se podrían utilizar sensores de cualquier tipo, desde sonares hasta *routers*, siempre que ayuden a conocer la posición del robot ya sea de manera precisa o aproximada. Del mismo modo, hemos utilizado el robot NAO, pero podríamos haber utilizado cualquier otro para realizar las pruebas.

7.2. Líneas futuras de actuación

En este trabajo, se han llevado a cabo diferentes pruebas utilizando un sistema multiagente, donde se han podido alcanzar los objetivos propuestos. Los resultados obtenidos han sido los esperados, por ese motivo, se puede continuar con el desarrollo de este trabajo de varias maneras. A continuación, se comentan algunas de las líneas de actuación que se podrían realizar:

- **Realizar las pruebas sobre un escenario más amplio.** El escenario que se ha utilizado era bastante reducido (2,25 m^2). Ampliando el escenario, se podrían realizar recorridos más complejos, haciéndolos más largos o, incluso, añadiendo obstáculos.

- **Utilización de diferentes dispositivos de sensorización en las pruebas.** En las pruebas realizadas, únicamente, se han utilizado cámaras, pero podrían utilizarse diferentes dispositivos de sensorización a la vez o por separado para estudiar el error cometido por el robot.

- **Utilización de diferentes robots en las pruebas.** Como en el punto anterior, también se podrían utilizar diferentes tipos de robots y estudiar el error cometido por éstos. Además, se podrían complicar las pruebas añadiendo diferentes robots al mismo tiempo y ver cómo interactúan para alcanzar sus posiciones deseadas o como se coordinan para completar las tareas.

- **Prueba en un escenario real.** Por último, las pruebas han sido en un entorno controlado, donde disponemos de un entorno estático que se conoce, pero sería interesante incorporar al robot en un entorno real donde interactuase en un entorno cambiante con la ayuda de los diferentes sensores disponibles en éste.

8. Referencias

[1] Yanahuaya Arce, A. (2015). Sistemas de vigilancia de personas mayores usando sistemas multiagente y robots bípedos controlados mediante voz.

[2] Fábregues de los Santos, L. (2015). Algoritmo distribuido para la asignación de tareas en un equipo de robots NAO.

[3] Martín Rico, F., Barrera González, P., Cañas, J. M., & Matellán Olivera, V. (2012). Localización topológica basada en visión par robots móviles.

[4] Ezquerra Cerdán, C. (2006). Auto localización de robots móviles y modelado del entorno mediante visión estereoscópica.

[5] Fernández Lorenzo, I. (2005). Sistema de posicionamiento absoluto de un robot móvil utilizando cámaras externas.

[6] Pizarro Pérez, D. (2008). Localización de robots móviles en espacios inteligentes utilizando cámaras externas y marcas naturales.

[7] Blanes, F., Muñoz, M., Simó, J., Martínez, H., José Alcaraz, J.: Arquitecturas de control sobre robots nao en la spl de robocup. In: III Workshop de Robótica: Robótica Experimental, ROBOT 2011, pp. 128–134. Universidad de Sevilla (2011) ISSN 978-84-615-6787-4

[8] Hípola, Pedro and Vargas-Quesada, Benjamín and Montes, Agustín Descripción y evaluación de agentes multibuscadores. *El profesional de la información*, 1999, vol. 8, n. 11. [Journal article (Print/Paginated)].

[9] Esmeralda Ramos y Jiménez Silvestre - *Agentes Inteligentes* - ND. 2000-01. Lecturas en Ciencias de la Computación ISSN 1316-6239. Escuela de Computación. Facultad de Ciencias. Universidad Central de Venezuela.

[10] Posadas Yagüe, J. L. (2015). Arquitectura para el control de robots móviles mediante delegación de código y agentes (Doctoral dissertation).

[11] Guifré Cuní, Marc Esteva, Pere Garcia, Eloi Puertas, Carles Sierra, and Teresa Solchaga. MASFIT: Multi-Agent System for Fish Trading. In In Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), València, Spain, 2004.

[12] Weiss, G. (1999). Multiagent systems: a modern approach to distributed artificial intelligence. MIT press.

[13] M. Guadalupe (2007). Arquitectura tolerante a fallos mediante un sistema multiagente para el sistema de control de un robot móvil.

[14] J. Ferber. Multi-Agent System: An Introduction to Distributed Artificial Intelligence. Addison Wesley, 1999.

[15] Foundation for Intelligent Physical Agent. Specification. http://www.fipa.org, 2004.

[16] A. Mas. Agentes software y sistemas multiagente, Conceptos, arquitecturas y aplicaciones. Pearson, 2004.

[17] N. Nilsson. Inteligencia Artificial: Nueva Síntesis. McGraw-Hill, 1964.

[18] N. Vlassis. A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. In R. Brachman and T. Dietterich, editors, Synthesis lectures on artificial intelligence and machine learning. Morgan and Claypool Publishers, 2008.

[19] M. Wooldridge. An Introduction to MultiAgent Systems. John Wiley & Sons, second edition edition, 2009.

[20] Vidal, J. Fundamentals of multiagent systems with netlogo examples. http://www.damas.ift.ulaval.ca/coursMAS/ComplementsH10/mas-Vidal.pdf, 2010.

[21] Gutknecht, O. and Ferber, J. Vers une méthodologie organisationnelle de conception de systèmes multi-agents. JFIADSMA'99, La Réunion, Hermès, pp. 93-104, 2000.

[22] Gutknecht, O. and Ferber, J. Madkit: a Generic Multi-Agent Platform (short paper). Autonomous Agents (AGENTS 2000), Barcelona, ACM Press, pp. 78-79, 2000.

[23] MADKit Multi-Agent Development Kit. www.madkit.org, 2002.

[24] Ferber J., Gutknecht O. Aalaadin: a meta-model for the analysis and design of organizations in multi-agent systems, ICMAS 98 (International Conference on Multi-Agent Systems), Paris, Y. Demazeau (ed), IEEE Press, pp. 128-135, 1998.

[25] F. Bellifemine, G. Caire, and D. Greenwood. Developing Multi-Agent Systems with JADE. John Wiley & Sons, 2007.

[26] K. Wang, W. Abdulla, and Z. Salcic. Ambient intelligence platform using multi-agent system and mobile ubiquitous hardware. Pervasive and Mobile Computing, 5(5):558– 573, 2009.

[27] JADE, Java agent development framework. In http://jade.tilab.com, 2014.

[28] *SPADE*: Smart Python multi-Agent Development Environment. http://gti-ia.dsic.upv.es/projects/magentix/.

[29] Gregori, M. E., Cámara, J. P., & Bada, G. A. (2006, May). A jabber-based multi-agent system platform. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems* (pp. 1282-1284). ACM.

[30] Luis Mingueza, N. (2013). Desarrollo de un sistema de juego al Tres en Raya para el robot NAO H25.

[31] J. Ruiz del Solar and R. Salazar. Introducción a la robótica. http://robotica.li2.uchile.cl/EL63G/

[32] Aldebaran Robotics 2014, Documentación y especificaciones del robot NAO, http://doc.aldebaran.com/1-14/dev/naoqi/index.html

[33] Michel, O. (1998, July). Webots: Symbiosis between virtual and real mobile robots. In *Virtual Worlds* (pp. 254-263). Springer Berlin Heidelberg.