Tesis de Maestría

PEDECIBA Informática

Contribución al diseño de sistemas multi robots utilizando ALLIANCE

Gonzalo Daniel Tejera López
gtejera@fing.edu.uy

Orientador: Dr. Juan Miguel Santos
Supervisor: MSc. Dina Wonsever

Instituto de Computación
Facultad de Ingeniería
Universidad de la Republica

Noviembe de 2004

Agradecimientos

Este trabajo comenzó como un sueño y hoy gracias al apoyo de mucha gente es una realidad. Quiero agradecer a Dina y a Gustavo por permitirme comenzar este trabajo de maestría, y por el apoyo aún hoy, luego de muchos años de comenzado. A Juan, quien sin conocerme me trato como si nos conociéramos de toda la vida, con el cual me siento muy cómodo y gracias a quien pude realizar este trabajo.

Quiero agradecer a todos los compañeros del Departamento de Arquitectura por el aguante día a día. Al Instituto de Computación por permitir continuar mi formación en temas no desarrollados en sus estructuras. Al PEDECIBA por apoyar económicamente mis estudios y a la CSIC por apoyar económicamente mis pasantías en Buenos Aires.

A Amparo, a mi familia y amigos, que me han apoyado durante este largo camino.

Resumen

Los sistemas multi robots permiten mejorar el rendimiento y la tolerancia a fallas respecto a los sistemas compuestos de un único robot monolítico. La arquitectura de control se encarga de lograr estos dos objetivos principales. ALLIANCE es una arquitectura de software de control atractiva e interesante si se tiene en cuenta su origen, su objetivo y su modelo formal. La misma fue concebida para asignar tareas a robots heterogéneos de manera robusta respecto a cambios inesperados en el entorno y a modificaciones que puedan ocurrir al equipo de robots.

Unos años más tarde aparece L-ALLIANCE, que además de conservar las propiedades de su predecesora mejora la selección cooperativa de acciones y aliviana notablemente la tarea de ajuste de parámetros requerida por ALLIANCE al diseñador del equipo de robots.

En este trabajo se estudia con detenimiento la arquitectura de control ALLIANCE y su sucesora L-ALLIANCE, realizando modificaciones en su modelo formal y proponiendo nuevas heurísticas que tratan de obtener una mejor asignación, redundando en un mejor rendimiento del equipo. Se presentan resultados que comparan la heurística propuesta en L- ALLIANCE con las propuestas en este trabajo utilizando el simulador de robots Khepera YAKS. El caso de estudio se centra en una tarea de recolección de objetos implementada utilizando máquinas de estado y comportamientos reactivos.

Se define un núcleo de ALLIANCE denominado N-ALLIANCE sobre el cual se implementa ALLIANCE, L-ALLIANCE y las nuevas propuestas. Este núcleo brinda mayor flexibilidad conservando al mismo tiempo las principales virtudes de ALLIANCE.

Palabras clave: sistemas multi robots, tolerancia a fallas, cooperación y asignación de tareas.

Índice general

Índice de tablas

Índice de algoritmos

Índice de figuras

1. Presentación

1.1. Introducción

Una arquitectura está descripta por un conjunto de componentes y cómo estos interactúan. En este trabajo se estudiaron las arquitecturas de control ALLIANCE [Parker1998] y L-ALLIANCE [Parker1995], las cuales permiten controlar un equipo de robots heterogéneos autónomos que deben realizar una determinada misión. De este estudio se obtiene un resumen claro de estas arquitecturas facilitando su uso a los diseñadores de equipos de robots.

Cuando se trabaja con un robot es muy común que falle alguno de sus componentes y que dicha falla inutilice totalmente el robot. La arquitectura ALLIANCE fue pensada para soportar las fallas que puedan ocurrir en los robots durante la misión, y por lo tanto cuando la arquitectura detecta esta situación adapta la asignación de tareas dinámicamente.

La arquitectura L-ALLIANCE fue propuesta por Parker [Parker1995] con los objetivos de facilitar la utilización de ALLIANCE y mejorar el desempeño del equipo. Estos objetivos están relacionados, pues ambos evitan el ajuste manual de gran cantidad de parámetros que debe realizar el diseñador del equipo de robots cuando utiliza la arquitectura ALLIANCE. Definir un mecanismo de actualización de parámetros automático permite que el diseñador se concentre en la formación del equipo de robots y los comportamientos individuales de cada uno de ellos. Para lograr esto Parker propone con L-ALLIANCE un mecanismo automático de actualización de parámetros y unas pequeñas modificaciones al formalismo de ALLIANCE.

Se implementó en Java la arquitectura ALLIANCE, lo cual permitió comprobar que la misma utilizaba un mecanismo simple, interesante y robusto para la asignación de tareas a robots, pero al momento de implementarla se vio que su formalismo tenía algunas inconsistencias respecto de las descripciones o intenciones por parte de su creador. También se observó que Parker modifica el formalismo de ALLIANCE para cada uno de los problemas que desea resolver, en lugar de utilizar ALLIANCE y construir sobre esta la solución al problema.

A partir de las observaciones antes mencionadas se desarrolló un núcleo de ALLIANCE (N-ALLIANCE), esto es, una definición que reduce al mínimo las ideas y bases sobre las cuales se diseño ALLIANCE (permitir construir equipos de robots tolerantes a fallas). Luego este núcleo se puede extender para construir sobre él ALLIANCE, L-ALLIANCE u otras arquitecturas que se deseen. Esto se verificó construyendo ALLIANCE y L-ALLIANCE.

Por otro lado, en este trabajo se proponen dos nuevas estrategias de asignación con el objetivo de mejorar el desempeño respecto de las propuestas de L-ALLIANCE. Una de estas estrategias pretende atacar el problema de tareas repetitivas el cual hasta el momento no había sido tenido en cuenta. Estas estrategias son implementadas sobre N-ALLIANCE y comparadas con L-ALLIANCE, utilizando una misión de recolección sobre el simulador YAKS [Zaxmy2003] y misiones ficticias generadas automáticamente.

El caso de estudio que se utiliza para realizar los experimentos usa los comportamientos “recolectar objetos” y “reportar actividad”. Dichos comportamientos de alto nivel se construyen con máquinas de estados y comportamientos reactivos.

1.2. Trabajos relacionados

Este nuevo enfoque de ALLIANCE, denominado N-ALLIANCE, y las dos nuevas heurísticas que obtienen sub-óptimos de asignación de tareas constituyen los principales aportes de esta tesis.

También se pretende estudiar las bondades y debilidades de ALLIANCE y L-ALLIANCE con el objetivo de facilitar su uso para el diseño de equipos de robots cooperativos. Excepto por los trabajos de Parker [Parker1995, Parker1999] no se han realizado otros estudios en este sentido. Si se puede encontrar en la bibliografía varios autores [Arkin1998, Cao1995, Gerkey2003, Murphy2000] que citan a los trabajos de Parker resaltando sus virtudes, pero nada relacionado a la utilización de las arquitecturas propuestas en casos reales ni proponiendo mejoras en su asignación.

Se investigaron varias arquitecturas de control entre las que se destacan las propuestas por David Jun [Jung1999] y Maja Mataric [Mataric1995]. La arquitectura propuesta por Jun no separa el comportamiento individual del comportamiento de grupo. Esta arquitectura está orientada a resolver problemas de colaboración de manera eficiente sin tener en cuenta ni la cooperación ni la tolerancia a fallas. El comportamiento de cada robot se construye como un grafo, donde los nodos son pequeñas unidades de comportamientos denominados módulos de capacidad (competence module) (CM) y detectores de características (feature detector) (FD). Sólo un CM puede estar activo en un momento dado. Las aristas se utilizan para definir precondiciones y correlaciones. Las precondiciones son características que deben cumplirse (FD) para activar un CM. Las correlaciones determinan que efecto tiene la activación de un CM sobre los FDs. Por otro lado Mataric presenta una metodología que utiliza comportamientos básicos para generar varios comportamientos de grupo robustos. En lugar de una prueba analítica, propone una evaluación empírica del rendimiento de cada algoritmo basándose en el siguiente criterio:

- Repetibilidad: cuán consistente es el comportamiento en diferentes corridas.
- Estabilidad: oscila el comportamiento bajo ciertas circunstancias.
- Robustez: cuán robusto es el comportamiento en presencia de errores y ruido en los sensores y actuadores.
- Escalabilidad: cómo es afectado el comportamiento del grupo con el incremento o decremento del tamaño del grupo.

Esta propuesta está motivada por entender y analizar los comportamientos sociales y de grupo en la naturaleza, y desarrollar una metodología para el diseño de comportamientos de grupo en sistemas artificiales. Trabaja con agentes homogéneos que no usan modelos explícitos de sus compañeros, no utilizan comunicación directa o cooperación explicita. Cao1995 se presenta el resumen de otras arquitecturas de control para sistemas multi robots.

1.3. Guía de Capítulos

En el Capítulo 2 se presenta una introducción a los robots comenzando con una reseña histórica, luego se definen y clasifican los robots en las siguientes categorías: robots industriales, robots de servicio y robots inteligentes. Finalmente se presentan los sistemas multi robots, sus características y el problema de asignación de tareas a robots.

En el Capítulo 3 se realiza un resumen de las arquitecturas propuestas por Parker, ALLIANCE [Parker1998] y L-ALLIANCE [Parker1995], incluyendo críticas al formalismo propuesto y a la manera en que estas arquitecturas son utilizadas por ella. También se muestran varios ejemplos que plantean problemas de rendimiento cuando se utiliza L-ALLIANCE.

En el Capítulo 4 se propone, justifica y formaliza N-ALLIANCE (el núcleo de ALLIANCE) a partir del cual luego se construyen las arquitecturas ALLIANCE y L-ALLIANCE.

En el Capítulo 5 se presenta nuevas estrategias para resolver el problema de asignación de tareas a robots tratando de mejorar el desempeño del equipo. La primera estrategia, denominada TC/N-ALLIANCE, toma en cuenta la heterogeneidad de comportamientos entre los distintos robots. La segunda estrategia, denominada CR/N-ALLIANCE, apunta a que los robots se dividan las tareas de forma uniforme de acuerdo al rendimiento esperado de ejecución.

En el Capítulo 6 se presentan el caso de estudio elegido (misión de recolección), y los resultados obtenidos con las diversas estrategias de control. También se realiza un análisis de los resultados a partir de la simulación de varias misiones variando la cantidad de robots y la cantidad de tareas a realizar.

En el Capítulo 7 se presentan los comportamientos de alto nivel utilizados en el caso de estudio. Estos comportamientos se construyen como máquinas de estado, en donde las transiciones se definen como predicados sobre el sistema sensorial del robot y los estados como comportamientos reactivos.

En el Capítulo 8 se presentan las conclusiones que se desprenden de este trabajo y las principales líneas de investigación para explorar a partir de lo realizado aquí.

2. Robots y Sistemas Multi-Robots

2.1. Introducción

En este capítulo se presenta un resumen de los temas abordados en este trabajo, estos son: los robots, los sistemas multi-robot y la asignación de tareas. No pretende realizar un estudio exhaustivo en ningún caso sino que sólo se pretende proporcionar conocimiento necesario a las personas que no se encuentren familiarizadas con estos temas. Si se desea profundizar en robótica se puede consultar la siguiente bibliografía: [Arkin1998], [Barrientos1997] y [Murphy2000].

En el Apéndice C se presentan varios robots actuales fabricados principalmente con el objetivo del entretenimiento.

2.2. Robots

La palabra robot puede generar miedo, rechazo y fascinación en las personas. Pero, ¿qué es un robot? Cuando se piensa en un robot se presenta a nuestra mente distintas imágenes entre las cuales se incluyen la un brazo mecánico trabajando de forma rutinaria sobre una cadena de montaje y el robot de ciencia-ficción con características y capacidades humanas. En este trabajo la palabra robot se refiere al estereotipo asociado en la literatura al robot inteligente, los cuales son capaces de resolver problemas en entornos dinámicos y peligrosos. Por otro lado existe el robot industrial, el cual es una especie de brazo mecánico animado que con rapidez y precisión suelda carrocerías de un vehículo o inserta circuitos integrados en placas electrónicas.

El mito ha rodeado y rodea al robot, a pesar que existen más de un millón de unidades de robots industriales operativas sólo en Japón sin contar los distintos robots inteligentes que han aparecido en el ámbito académico en estos últimos años. Sus orígenes de ficción, su controvertido impacto social, su aparente autonomía y notorio contenido tecnológico origina que, a pesar de su popularidad, siga siendo admirado y en ocasiones temido.

La robótica posee un reconocido carácter interdisciplinario, participando en ella disciplinas básicas y tecnológicas tales como la teoría de control, la mecánica, la electrónica, la informática, la psicología y la etología entre otras.

2.2.1. Reseña histórica

A lo largo de toda la historia, el hombre se ha sentido fascinado por máquinas y dispositivos capaces de imitar las funciones de los seres vivos. Algunos de estos autómatas se enumeran a continuación:

- La estatua de Memon de Amenhotep (1500 a.c.) que emite sonidos cuando la iluminan los rayos del sol.
- La urraca voladora y el caballo que salta de King-su Tse (500 a.c.).
- La cabeza parlante de Roger Bacon (1214-1294).
- El león mecánico de Leonardo Da Vinci (1452-1519).

Durante los siglos XVII y XVIII se crearon ingenios mecánicos que tenían alguna de las características de los robots actuales. Estos dispositivos fueron creados en su mayoría por artesanos de la relojería y representaban figuras humanas o animales.

Los autómatas construidos hasta ese entonces no tenían una aplicación práctica y solamente servían para entretener. Algunos de estos mecanismos funcionaban por medio de movimientos ascendentes de agua o aire pero también podían encontrarse otros que utilizan palancas o contrapesos.

Tras los primeros autómatas descriptos, algunos de aspecto humano, los progenitores más directos de los robots fueron los telemanipuladores. En la Figura 7 se puede apreciar el manipulador desarrollado por la empresa Central Research Laboratorios (CRL). En 1948 R.C. Goertz del Argonne National Laboratory desarrolló el primer telemanipulador, con el objetivo de manipular elementos radiactivos sin riesgo para el operador. El telemanipulador consistía en un dispositivo mecánico maestro-esclavo. El manipulador maestro, situado en la zona segura, era controlado directamente por el operador, mientras que el esclavo, situado en contacto con los elementos radiactivos y unido mecánicamente con al maestro, reproducía fielmente los movimientos de éste. El operador además de poder observar a través de un grueso cristal el resultado de sus acciones, sentía a través del dispositivo maestro, las fuerzas que el esclavo ejercía sobre el entorno.

Figura 1. Telemanipulador fabricado por CRL [CRL2004].

El siguiente paso fue sustituir al operador del manipulador maestro y al manipulador maestro por un programa de ordenador que controlase los movimientos del manipulador. Este avance dio lugar al concepto de robot industrial desarrollado por George Devol y Joe Engelberger a comienzos de la década del 60. En pequeñas o grandes fábricas, estos robots pueden sustituir al hombre en aquellas tareas repetitivas y peligrosas, adaptándose inmediatamente a los cambios de producción.

Desde el punto de vista de la ciencia-ficción, la palabra robot fue usada por primera vez en el año 1921, cuando el escritor checo Karel Capek (1890-1938) estrena en teatro nacional de Praga su obra Rossum's Universal Robot. Su origen es la palabra eslava robota, que se refiere al trabajo realizado de manera forzada. En la obra los robots eran máquinas androides fabricadas a partir de la formula obtenida de un brillante científico llamado Rossum. Estos robots servían a sus jefes humanos desarrollando trabajos forzados, hasta que finalmente se rebelan contra su creador.

El término posiblemente hubiera caído en desuso si no hubiera sido por los escritores del género literario de la ciencia-ficción y el cine. El máximo impulsor de la palabra robot fue el escritor norteamericano de origen ruso Isaac Asimov (1920-1992), que en 1945 publicó en la revista Galaxy Science Fiction una historia en la que enunció por primera vez sus tres leyes de la robótica[1], estas leyes se grababan en los cerebros positrónicos de los robots. A continuación se presentan las tres leyes de la robótica propuestas por Asimov:

1. Un robot no puede perjudicar a un ser humano, ni con su inacción permitir que un ser humano sufra daño.
2. Un robot ha de obedecer las órdenes recibidas de un ser humano, excepto si tales órdenes entran en conflicto con la primera ley.
3. Un Robot debe proteger su propia existencia mientras tal protección no entre en conflicto con la primera o segunda ley.

Asimov se atribuye la creación del término robotics (robótica) y sin lugar a duda, desde su obra literaria[2], ha contribuido decisivamente a la divulgación y difusión de la robótica.

Por otro lado, existen varios ejemplos en la literatura y en el cine que muestra lo peligrosas que pueden ser estas máquinas inteligentes para el hombre. El libro DUNE [Herbert1984] es un claro ejemplo literario que propone a los robots como problemáticos aun cuando este no es el tema central del libro, pasando desapercibido entre los lectores.

Los futuros de la robótica apuntan a aumentar su movilidad, destreza y autonomía de acción. La mayor parte de los robots actuales son con base estática, y se utilizan en aplicaciones industriales tales como ensamblado y soldadura. Sin embargo, existen otras aplicaciones que han hecho evolucionar en gran medida tanto la concepción de los robots como su propia morfología. Con este fin se desarrollan los robots de servicio dedicados a aplicaciones no industriales, donde se destacan los robots espaciales, robots para aplicaciones submarinas y subterráneas, robots militares, robots móviles industriales y aplicaciones médicas. También se está trabajando en robots inteligentes, construyendo criaturas artificiales que se mueven en el mundo con un determinado objetivo de manera autónoma.

2.2.2. Definiciones

Buena parte de las definiciones y clasificaciones de robots existentes responde al robot ampliamente utilizado hasta la fecha, destinado a la fabricación flexible en líneas de montaje en serie y que se conoce como robot industrial o robot de producción. Frente a estos, los robots especiales o de servicio, están aún en un estado de desarrollo incipiente, aunque es previsible un considerable desarrollo de los mismos.

---

[1] En posteriores publicaciones el autor agrega una cuarta ley.

[2] Se recomienda la lectura de libro Yo Robot [Asimov1950].

2.2.2.1 Robots Industriales

La definición más aceptada posiblemente sea la de la Asociación de Industrias Robóticas (Robotic Industries Association) [RIA2004], según la cual:

> Un robot industrial es un manipulador multifuncional programable, capaz de mover materias, piezas, herramientas o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas.

Esta definición, ligeramente modificada ha sido adoptada por la Organización Internacional de Estándares (International Standard Organisation) [ISO1994] que define al robot industrial como:

> Manipulador multifuncional reprogramable con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas.

Una definición más completa es la establecida por la Asociación Francesa de Normalización (Association Française de Normalisation) [AFNOR2004] que define primero el manipulador y luego el robot:

> Manipulador: mecanismo formado generalmente por elementos en serie, articulados entre sí, destinado al agarre y desplazamiento de objetos. Es multifuncional y puede ser gobernado directamente por un operador humano o mediante dispositivo lógico.
>
> Robot industrial: manipulador automático servocontrolado, reprogramable, polivalente, capaz de posicionar y ordenar piezas, útiles o dispositivos especiales, siguiendo trayectorias variables reprogramables, para la ejecución de tareas variadas. Normalmente tiene la forma de uno o más brazos terminados en una muñeca. Su unidad de control incluye un dispositivo de memoria y ocasionalmente de percepción del entorno. Normalmente se utilizan para realizar una tarea de manera cíclica, pudiéndose adaptar a otra sin cambios permanentes en su material.

Por último, la Federación Internacional de Robótica (International Federation of Robotics) [IFR2004] distingue entre robots industriales de manipulación y otros robots (en la siguiente sección):

> Por robot industrial de manipulación se entiende a una máquina de manipulación automática reprogramable y multifuncional con tres o más ejes que pueden posicionar y orientar materias, piezas, herramientas o dispositivos especiales para la ejecución de trabajos diversos en las diferentes etapas de la producción industrial, ya sea en una posición fija o en movimiento.

2.2.2.2 Robot de servicio

Los robots de servicio se han desarrollado en paralelo con los robots industriales aunque de manera más lenta. Una definición de robots de servicios según [Barrientos1997] es la siguiente:

> Dispositivos electromecánicos móviles o estacionarios, dotados normalmente de uno o varios brazos mecánicos independientes, controlados por un programa de ordenador y que realizan tareas no industriales de servicio.

La IFR ha adoptado la siguiente definición:

> Robot que opera de forma parcial o totalmente autónoma para realizar servicios útiles para el bienestar de los humanos y del equipamiento, excluyendo operaciones de manufactura.

Dentro de esta familia de robots se encuentran a los robots dedicados a cuidados médicos, educación, domésticos, uso en oficinas, intervención en ambientes peligrosos, aplicaciones espaciales, aplicaciones submarinas y agricultura.

2.2.2.3 Robot inteligente

Las definiciones de robots dadas hasta el momento fueron redactadas por organismos internacionales de normas. Otras definiciones redactadas por investigadores, no tan orientadas a la robótica industrial, pueden incluir términos como aprendizaje e inteligencia.

Para Ronald Arkin [Arkin1998] un robot inteligente es una máquina capaz de extraer información de su entorno y utilizar su conocimiento del mundo para moverse de forma segura cumpliendo un propósito y sentido.

Para Robin Murphy [Murphy2000] un robot inteligente es una criatura mecánica capaz de funcionar de manera autónoma.

El área de Aprendizaje en Robots (Robot Learning) estudia la modificación automática del comportamiento del robot de modo de mejorar su rendimiento a lo largo del tiempo. Según Claude Touzet [Touzet1998] aprendizaje en robots estudia la selección de un comportamiento eficiente del conjunto de comportamientos potenciales.

En el contexto de este trabajo se utilizará la definición propuesta por Murphy.

2.2.3. Sensores y actuadores

Los robots disponen de sensores y actuadores para interactuar con el entorno. Los actuadores le permiten al robot efectuar acciones y los sensores le permiten percibir la situación actual del entorno. Existen varios tipos de sensores entre los cuales se distinguen los siguientes: infrarrojo, ultrasonido, temperatura, cámara de video y GPS. Existen también varios tipos de actuadores, como ser: articulaciones prismáticas (estas se acortan o se alargan) o rotacionales formando patas o brazos, ruedas, orugas, en general movidas por fuerza eléctrica, neumática o hidráulica, usando motores, servos o válvulas. Muchas veces para referirse las características de movilidad de determinados actuadores se utiliza el término grados de libertad, que define el número de formas diferentes que se puede mover una parte del robot.

2.2.4. Un ejemplo: el robot Khepera

Luego de esta introducción a los robots se presenta en este punto el pequeño robot Khepera. Durante una pasantía en el Departamento de Computación de la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires se trabajó con un robot Khepera en síntesis de comportamientos, sumado a su popularidad en diversas universidades y utilización en varios trabajos de investigación, se decidió incluirlo como ejemplo de robot y probar los resultados de este trabajo en él.

El robot Khepera [K-Team1999a] fue desarrollado en Suiza para ser utilizado en investigación y desarrollo. En la Figura 2 se muestra el robot Khepera, en la misma se pueden apreciar las dimensiones y la apariencia del mismo.

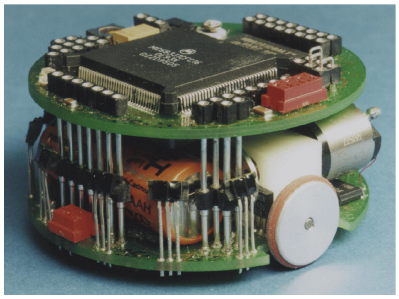


Figura 2. Robot Khepera [Touzet1998].

En la Tabla 1 aparecen las características técnicas del robot Khepera. Dispone de 8 sensores infrarrojos distribuidos alrededor de su estructura. Es posible consultar el valor de estos sensores en todo momento y el valor retornado es un número entero sin signo de 10 bits. Un valor alto indica la presencia de un objeto próximo a sensor consultado, por otro lado, un valor bajo indica que el sensor consultado no detecta un objeto próximo a él. Estos sensores permiten detectar objetos a no más de 5 cm.

Tabla 1. Características técnicas del robot Khepera.

Elemento	Descripción
Procesador	Motorola 68331
RAM	256 KB
ROM	256 o 512 KB
Actuadores	2 motores DC con codificador de incremento
Sensores	8 sensores infrarrojos de proximidad y luz
Energía	Externa o batería de NiCd recargable.
Autonomía	30 minutos
Tamaño	55 mm de diámetro y 30 mm de altura
Peso	Aproximadamente 70 grs.

Para moverse en el entorno el robot dispone de dos motores de corriente continua (DC). Es posible especificar de forma independiente la velocidad de cada motor indicando un valor entre -127 y 127. Si se le indica como velocidad 127 en ambas ruedas el robot avanzará a una velocidad aproximada de 1 m/s.

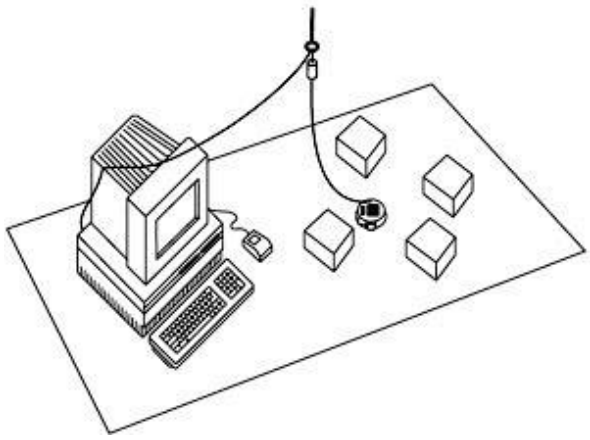


Figura 3. Entorno de trabajo habitual para el robot Khepera [K-Team1999a].

Existen dos formas para trabajar con el robot Khepera. Una de ellas, y más habitual, es enviarle comandos dinámicamente utilizando un cable que conecta el robot al puerto serial de la computadora. La otra forma es descargar el programa en el robot para que luego este lo ejecute y se mueva de forma autónoma en el entorno de acuerdo al programa descargado. En la Figura 3 se ilustra la forma habitual de trabajo. La conexión entre la computadora y el robot permite enviarle comandos dinámicamente o enviarle el programa que debe ejecutar. Este cable puede sustituirse, en caso de disponer del hardware de comunicación adecuado, por un enlace inalámbrico [K-Team1999b, K-Team1999c].

En el Capítulo 7 se muestran algunas pruebas realizadas con un simulador del robot Khepera.

2.3. Sistemas multi-robot

2.3.1. Introducción

Los robots pueden utilizarse en la industria para sustituir al hombre en tareas peligrosas, pesadas, cansadoras o aburridas. Pero este no es el único uso ni lugar posible para los robots. En los últimos años los robots comenzaron a aparecer en las universidades para la investigación y el trabajo con robots móviles autónomos. El objetivo es que estos robots puedan moverse en el mundo de forma autónoma con un propósito determinado, como ser apagar incendios, vigilar una zona, recolectar rocas, limpiar desechos tóxicos, etc. Estos robots son programados para que puedan desarrollar sus tareas sin la necesidad de presencia humana, para lo cual son dotados con los sensores y actuadores requeridos para la tarea que se les encomienda. El robot realiza la tarea obteniendo información del entorno mediante sus sensores y controlando sus actuadores de acuerdo al comportamiento programado.

Generalmente los robots tienen un componente de software y otro de hardware, y como es de esperar pueden ocurrir fallas tanto en uno como en otro. Además es común que una falla en un componente del robot determine la falla del robot. Si se desea que en el futuro los robots resuelvan tareas críticas no se puede permitir que una falla en alguno de sus componentes lleve a un fracaso en la tarea asignada. Una solución a este problema es agregar componentes redundantes, aumentando de esta forma la tolerancia a fallas (en [Cavallaro1994] se detalla una serie de estándares aplicables y adaptables a la robótica con el objetivo de mejorar la confiabilidad).

Además, al utilizar un único robot para resolver la tarea obliga a que éste pueda resolver todas las sub-tareas requeridas, obteniendo un robot muy complejo que como explica [Jung1998] redunda en una menor confiabilidad. Además cuanto más confiables son los componentes más costosos son.

Otra solución es resolver la tarea utilizando un equipo de robots, donde la redundancia aparece al solaparse las capacidades de los distintos miembros del equipo.

[Cao1997] indica que los sistemas multi-robot son de interés por las siguientes razones:

- La tarea a realizar puede ser muy compleja (o imposible) para ser realizada por un único robot o pueden obtenerse beneficios importantes al utilizar múltiples robots.
- Construir y utilizar varios robots simples puede ser más barato, más flexible y más tolerante a fallas que tener un poderoso y único robot.
- El acercamiento constructivo, sintético inherente en robótica móvil cooperativa puede tratar problemas fundamentales en las ciencias sociales y ciencias de la vida. Se refiere a la posibilidad, mediante la construcción de robots, de verificar o tratar problemas en dichas ciencias.

Por las ventajas indicadas, gran número de soluciones humanas a las aplicaciones de interés comentadas emplean varias personas apoyándose, ayudándose y complementándose entre ellas. En lugar de tener una única persona realizando la tarea, se forman grupos de obreros para tener variedad de niveles de especialización. Estos obreros están disponibles para ayudarse mutuamente, y cuando la aplicación lo requiera, proporcionar experiencia individual. A cada miembro del grupo se le asigna un rol que realizará durante la misión el cual se basa en el nivel y la experiencia de la persona. A su vez, las personas comparten capacidades que permiten realizar tareas intercambiablemente, dependiendo de la carga de trabajo del individuo durante la misión. Pueden ocurrir eventos inesperados durante la misión que requieran una reubicación dinámica de tareas para enfrentar las circunstancias actuales.

Cuando se trabaja con sistemas multi-robots aparecen problemas que no aparecen cuando se trabaja con un único robot, algunos de estos problemas son:

- ¿Cómo formular, describir, descomponer, y asignar tareas entre un grupo de agentes distribuidos?
- ¿Cómo se permite que los agentes se comuniquen e interactúen?
- ¿Cómo se asegura que los agentes actúan coherentemente?
- ¿Cómo permitimos que los agentes reconozcan y resuelvan conflictos?

Cuando se trabaja con sistemas multi-robots aparece de forma inmediata el término cooperación. La Real Academia Española define cooperación como “acción y efecto de obrar juntamente con otro u otros para un mismo fin”. En los siguientes párrafos se introducen y definen los conceptos más importantes.

El término comportamiento colectivo denota cualquier comportamiento de agentes en un sistema formado por más de un agente. Mientras que comportamiento cooperativo es una subclase del comportamiento colectivo que se caracteriza por la cooperación.

A su vez los investigadores distinguen dos tipos de cooperación, la robótica colectiva y la robótica cooperativa. En ambos casos se desea obtener un comportamiento de grupo a partir de los comportamientos individuales. La robótica colectiva, también conocida como swarm robotics, puede caracterizarse por el control distribuido de

robots homogéneos. En este caso la dinámica colectiva deseada se obtiene como una propiedad emergente de comportamientos simples y la interacción local entre robots. Existen varios ejemplos de comportamiento colectivo en los insectos, como ser buscar alimento o construir nidos. Por otro lado, la robótica cooperativa usualmente involucra robots heterogéneos donde la dinámica cooperativa se alcanza utilizando la arquitectura de control.

En los distintos trabajos relacionados con la robótica cooperativa aparecen definiciones de cooperación. [Cao1997] define comportamiento cooperativo de la siguiente manera: “Dada una tarea especificada por el diseñador, un sistema multi-robots demuestra un comportamiento cooperativo si, debido a un mecanismo subyacente, hay un incremento en la utilidad total del sistema”. En [Bergfeldt2000] y [Cao1997] aparecen otras definiciones tomadas de los distintos artículos del área. En estas definiciones aparecen tres conceptos fundamentales: la tarea, el mecanismo de cooperación y el rendimiento del sistema.

Según [Cao1997] pueden identificarse cinco ejes de investigación: arquitectura de grupo, conflicto de recursos, origen de la cooperación, aprendizaje y problemas geométricos. Por ser de gran importancia para este trabajo solo se profundiza en la arquitectura de grupo.

2.3.2. Arquitectura de grupo

La arquitectura de grupo proporciona la infraestructura sobre la cual se implementan los comportamientos colectivos, y determina las capacidades y limitaciones del sistema. A continuación se describen las características principales de las arquitecturas de grupo para robots móviles.

2.3.2.1 Centralizado vs Descentralizado

La decisión fundamental a tomar, al momento de definir la arquitectura, es determinar si el control será centralizado o descentralizado, y en caso de ser descentralizado si será distribuido o jerárquico. Las arquitecturas centralizadas se caracterizan por tener un único agente de control. Las arquitecturas descentralizadas carecen de este agente.

Existen dos tipos de arquitecturas descentralizadas: arquitecturas distribuidas donde todos los agentes son idénticos respecto al control, y arquitecturas jerárquicas donde el control es localmente centralizado, respecto a un grupo de robots. El comportamiento de los sistemas descentralizados es a menudo descrito en términos de emergencia o auto-organización (más adelante se presenta la arquitectura ALLIANCE, la cual es una excepción).

El paradigma dominante es el descentralizado, existiendo un consenso respecto a las bondades de las arquitecturas descentralizadas sobre las centralizadas tales como: tolerancia a fallas, explotación natural del paralelismo, confiabilidad y escalabilidad.

2.3.2.2 Diferenciación

Se dice que un grupo de robots es homogéneo si las capacidades de los robots son idénticas, y heterogéneo si difieren. En general, los grupos heterogéneos dificultan la asignación de robots a tareas. En lugar de clasificar a los grupos de robots en heterogéneos u homogéneo [Balch2000] propone una métrica para la diversidad que brinda mayor información acerca de los grupos heterogéneos. Esta métrica se basa en la propuesta de Shannon acerca de la entropía de la información.

2.3.2.3 Estructura de comunicación

La estructura de comunicación del grupo determina los posibles modos de interacción entre los robots del grupo. Se distinguen tres tipos de interacciones que pueden soportarse en la arquitectura.

Interacción vía el entorno

El tipo más simple de interacción ocurre cuando el medio de comunicación es el propio entorno y no existe comunicación o interacción entre los agentes. Esta modalidad es también denominada "cooperación sin comunicación". Existen varios trabajos que utilizan el entorno como medio de comunicación generalmente relacionados con la robótica colectiva.

Interacción vía percepción

La interacción vía percepción permite que ocurran interacciones locales entre agentes como resultado de la percepción, sin comunicación explícita. Este tipo de interacción requiere que los agentes puedan distinguir a sus compañeros[3].

Interacción vía comunicación

Esta forma de interacción requiere comunicación explícita entre los agentes, mediante mensajes entre individuos (multicast) o todos (broadcast). Como se verá más adelante ALLIANCE y sus sucesoras utilizan este mecanismo de comunicación pero solamente por la capacidad limitada de los sensores.

2.3.2.4 Modelado de otros agentes

Modelar las intenciones, creencias, acciones, capacidades, o el estado de otros agentes puede llevar a una mejor colaboración entre los agentes. [Mataric1992] divide la noción de reconocer otros robots en tres categorías:

- Ignorar la coexistencia – el robot piensa que es el único robot en el entorno tratando a los demás robots como obstáculos.
- Coexistencia informada – los robots tienen la habilidad de percibir la presencia de los otros robots.
- Coexistencia inteligente – además de percibir la presencia de los robots se permite modelar los robots que estén dentro de un determinado radio.

2.4. El Problema de Asignación de Tareas a Robots

Cuando se trabaja con un sistema multi-robot inevitablemente el diseñador se enfrenta al problema de asignar a cada robot una tarea útil a realizar. Esta decisión se toma de modo de obtener un beneficio para el equipo, ya sea una mejora en rendimiento, una mayor confiabilidad o permitir realizar tareas que de otro modo no se podrían realizar. Este problema se conoce en la literatura como asignación de tareas a múltiples robots (Multi-Robot Task Allocation) (MRTA). Las arquitecturas de control son responsables de determinar que tarea ejecuta cada robot.

En la literatura se pueden encontrar varios intentos de formalizar el problema de asignación de tareas como un problema de optimización.

---

[3] En algunos artículos está habilidad la denominan "kin recognition".

[Gerkey2003] define el problema MRTA de la siguiente manera:

> Sean *m* robots, cada uno capaz de ejecutar una tarea a la vez; y *n* tareas ponderadas, cada una requiriendo un único robot para ejecutarla. Dado para cada robot un estimador de la eficiencia en ejecutar cada una de las tareas (cero si no puede ejecutar una determinada tarea), el objetivo es asignar tareas a robots de modo de maximizar eficiencia del sistema, teniendo en cuenta las prioridades de las tareas y los valores de eficiencia de los robots.

Esta definición obliga a que las tareas sean ejecutadas por un único robot, lo cual simplifica notablemente el problema pero puede llevar a un menor rendimiento.

El MRTA está sumamente emparentado al problema de asignación óptima (Optimal Assignment Problem) (OAP) y planificación (scheduling), en estos problemas se utilizan los términos máquina o procesador en lugar de robot y trabajo o programa en lugar de tarea.

A continuación se define formalmente el problema MRTA:

Dados

- un conjunto de *n* robots, denotadas $I_1, ..., I_n$
- un conjunto de *m* tareas ponderadas, denotadas $J_1, ..., J_m$ y sus ponderaciones $w_1, ..., w_m$
- U_{ij}, la utilidad del robot i respecto de la tarea j, $1 \le i \le n$, $1 \le j \le m$

Se desea determinar los α_{ij} que maximicen la función:

- $\sum_{i,j} \alpha_{ij} U_{ij} w_j$

Sujeto a

- $\alpha_{ij} \in \{0,1\}, \forall i, j$
- $\sum_{i} \alpha_{ij} \le 1, \forall j$ [4]
- $\sum_{j} \alpha_{ij} \le 1, \forall i$ [5]

Este problema se puede ver desde la perspectiva de los algoritmos de scheduling, en particular, [Gerkey2003] rescribe MRTA como un problema de scheduling que es NP-difícil, por lo que el problema de MRTA también lo es.

[4] En la definición original la sumatoria debe ser igual a uno. La modificación realizada permite tener tareas sin asignar.

[5] En la definición original la sumatoria debe ser igual a uno. La modificación realizada permite tener robots ociosos.

[Parker1995] propone un problema similar al MRTA que denomina Problema de la Eficiencia de ALLIANCE (ALLIANCE Efficiency Problem) (AEP). El mismo se define de la siguiente manera:

Dados

- $R = \{r_1, r_2, ..., r_r\}$ el conjunto de robots que conforman el equipo,
- $T = \{task_1, task_2, ...task_m\}$ el conjunto de tareas que conforman la misión,
- $q_i : T \to \Re^+$ [6] la función que determina la calidad con que el robot r_i realiza una determinada tarea y
- C_i *conjunto de tareas que puede realizar el robot* r_i.

Se desea determinar

- $U_i = \{task_j \,|\, el\ robot\ r_i\ activa\ la\ tarea\ task_j\ durante\ la\ misión\}, \forall i \in \{0,..,n\}$.

Que minimice

- $\max \left\{ \sum_{task_j \in U_i} q(task_j) \,\middle|\, i \in \{1,2,...,n\} \right\}$.

Sujeto a

- $C_i \subseteq T, \forall i \in \{1,...,n\}$ y
- $U_i \subseteq C_i, \forall i \in \{1,...,n\}$.

Nuevamente, este problema de asignación es NP-difícil [Parker1998].

Al igual que antes, esta definición no toma en cuenta la posibilidad de seleccionar varias veces una tarea durante una misma misión.

Como se verá en el siguiente capítulo estas definiciones simplifican de sobremanera el problema de asignación de tareas a robots durante una misión en el mundo real. No toman en cuenta la dinámica del entorno y por ello no realizan reasignación durante la misión, algo necesario en aplicaciones reales. Si se toman en cuenta los cambios que puedan ocurrir durante la misión la definición del problema es más compleja y se puede reducir a las anteriores, por lo que también será un problema NP-difícil.

[6] Cuanto menor sea el valor de *q* mayor es la calidad para realizar la tarea. En general, se utiliza como función q la duración de la tarea.

3. ALLIANCE y L-ALLIANCE

3.1. Introducción

En este capítulo se presentan las arquitecturas ALLIANCE y L-ALLIANCE. Además, conjuntamente a la descripción del formalismo, se realiza un análisis del mismo para facilitar la comprensión de algunos puntos que podrán resultar ambiguos a partir de la bibliografía en el tema. El objetivo de este capítulo es facilitar al diseñador de equipos de robots el uso de ALLIANCE como arquitectura de control.

Se realiza una breve crítica a la forma en que es usada la arquitectura ALLIANCE para resolver problemas reales, la forma en que es usada por L-ALLIANCE y a las limitaciones impuestas innecesariamente sobre ambas arquitecturas. Estas observaciones son el punto de partida para el desarrollo de la arquitectura N-ALLIANCE definida en el Capítulo 4.

Además, se proponen algunos ejemplos donde L-ALLIANCE presenta dificultades para obtener la mejor asignación de tareas.

3.2. ALLIANCE

3.2.1. Introducción

ALLIANCE nace de los estudios de doctorado de Lynne Parker en el año 1995 y aparece publicada como artículo en 1998 [Parker1998].

ALLIANCE es una arquitectura de software que facilita el control cooperativo tolerante a fallas de grupos de robots móviles heterogéneos realizando una misión compuesta de varias tareas débilmente acopladas que pueden tener dependencias de orden. La dependencia en el orden de las tareas no está definida ni modelada dentro de ALLIANCE, la misma debe imponerse utilizando sensores virtuales u otros mecanismos fuera de la arquitectura.

ALLIANCE es una arquitectura distribuida basada en comportamiento, que incorpora el uso de un modelo matemático de motivaciones (impaciencia y asentimiento) en cada robot para lograr la selección de acciones adaptativa.

Como los grupos cooperativos de robots trabajan en entornos dinámicos e impredecibles, ALLIANCE permite que los miembros de un grupo de robots respondan de forma robusta, confiable, flexible, y coherente con los cambios inesperados del entorno y las modificaciones en el grupo de robots que se pueden deber a fallas mecánicas. La arquitectura ALLIANCE también toma en cuenta, aprendizaje de un nuevo comportamiento por parte de un miembro del equipo y la modificación de la cantidad de miembros del grupo por parte de un agente exterior, al momento de realizar la asignación.

Tolerancia a fallas en este contexto se refiere a la habilidad del grupo de robots a responder a fallas de robots individuales o fallas en la comunicación que pueden ocurrir en cualquier momento durante la misión. La respuesta tolerante a fallas tratada en ALLIANCE, permite la re-asignación dinámica de tareas por parte de un miembro del equipo debido a fallas de un robot o cambios en el entorno que no permitan que un robot se desempeñe adecuadamente en una tarea.

Adaptabilidad es la habilidad del grupo de robots de modificar su comportamiento en el tiempo en respuesta a la dinámica del entorno, cambios en la misión del equipo, o cambios en las capacidades del grupo o composición, para mejorar el rendimiento o

prevenir la degradación del rendimiento. Adaptabilidad en un grupo cooperativo permite al grupo ser responsable de los cambios de rendimiento en un robot individual, a los cambios en el entorno, y a los cambios en la composición del grupo de robots (se agregan o se retiran miembros).

En este contexto se dice que el sistema es confiable si funciona correctamente cada vez que es utilizado.

ALLIANCE asume lo siguiente:

- Los robots en el grupo pueden detectar los efectos de sus propias acciones, con probabilidad mayor que cero.
- Un robot puede detectar las acciones de otro miembro del grupo con las mismas capacidades, con probabilidad mayor que cero. Estas acciones pueden detectarse de alguna forma, incluso mediante un mensaje de broadcast.
- Los robots en el grupo no mienten y no son intencionalmente adversarios.
- No se asegura que el medio de comunicación esté disponible.
- Los robots no poseen sensores y ni actuadores perfectos.
- Cualquier robot del grupo puede fallar, con probabilidad mayor que cero.
- Si un robot falla, puede no alcanzar a comunicar la falla a sus compañeros.
- No se dispone de almacenamiento centralizado (de conocimiento) del mundo.

En ALLIANCE no se necesita que los robots estén capacitados para reconocer las acciones de un compañero mediante la observación pasiva (interacción vía percepción), la cual puede ser muy difícil de alcanzar con las capacidades tecnológicas actuales de los sensores. Esto no determina que cuando avance la tecnología de percepción este mecanismo pueda ser reemplazado por interacción vía percepción. En lugar de ello, los robots determinan las acciones de sus compañeros mediante un mecanismo de comunicación explícito (interacción vía comunicación), donde los robots transmiten la información de sus actividades corrientes al resto del equipo, realizando un broadcast de dicha actividad, este mensaje puede ser escuchado o ignorado por el resto del equipo.

Cada robot en ALLIANCE dispone de varios conjuntos de comportamientos (behavior sets). Cada uno de estos conjuntos permite resolver alguna de las tareas requeridas por la misión. Parker denomina a este comportamiento de alto nivel conjunto de comportamientos o función de alto nivel, el cual construye basándose en la arquitectura Subsumption propuesta por Brooks [Brooks1986]. El conjunto de comportamientos de alto nivel determina entonces las capacidades de cada robot, esto es, qué tareas pertenecientes a la misión puede realizar.

ALLIANCE incluye también un conjunto de comportamientos de bajo nivel, asociados a mecanismos de supervivencia o reflejos, que pueden inhibir a los comportamientos de alto nivel y viceversa. Un ejemplo típico de comportamiento de bajo nivel es evitar obstáculos.

Luego, cada robot debe seleccionar individualmente las acciones apropiadas. Esto es, qué comportamiento de alto nivel debe activar durante la misión, basado en los requerimientos de la misión, las actividades de otros robots, las condiciones actuales del entorno, y de su estado actual. Para esto, ALLIANCE recurre a un mecanismo basado en un modelo matemático que toma en cuenta dos motivaciones presentes en cada robot: impaciencia y asentimiento. Esto permite alcanzar una selección de tareas adaptable.

En ALLIANCE sólo un comportamiento de alto nivel puede estar activo en un momento dado mientras que el resto de los comportamientos dentro de robot deben estar hibernando. Esto es implementado mediante inhibiciones cruzadas (cross-inhibition) entre los comportamientos motivacionales. La selección del comportamiento activo es realizada mediante el uso de comportamientos motivacionales (motivational behavior), cada uno de los cuales controla la activación de un conjunto de comportamientos. Sin embargo, otros comportamientos de bajo nivel como ser evitar obstáculos deben estar continuamente activos sin importar el objetivo de alto nivel que este realizando actualmente el robot.

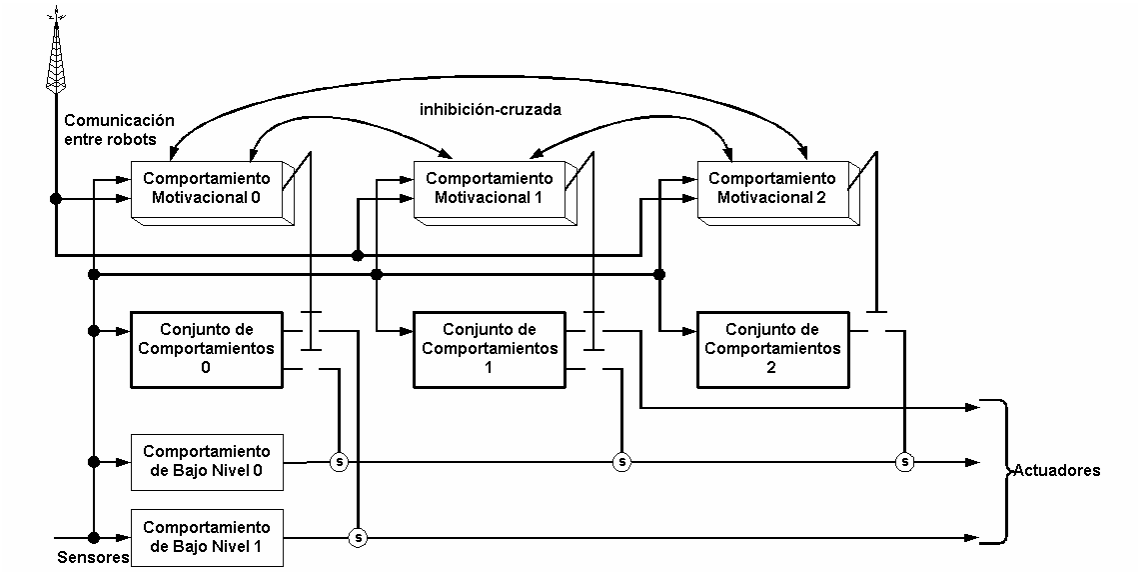


Figura 4. Diagrama de la arquitectura ALLIANCE.

En la Figura 4 se puede apreciar un diagrama de la arquitectura ALLIANCE instanciada en un determinado robot. Cada comportamiento motivacional recibe la lectura de los sensores, los mensajes transmitidos y la información de si otro conjunto de comportamientos está activo, y determina si debe activar el comportamiento de alto nivel asociado a él. También se muestran los comportamientos de bajo nivel asociados generalmente a mecanismos de supervivencia.

La arquitectura ALLIANCE no se preocupa de cómo se definen ni de qué tarea realiza los comportamientos de alto nivel sino del mecanismo de selección de estos.

3.2.2. Comportamientos motivacionales

En ALLIANCE, la habilidad en los robots de responder a eventos inesperados, fallas de robots, y demás, es alcanzada mediante el uso de motivaciones. Estas motivaciones están diseñadas para permitir a los miembros del grupo de robots realizar tareas sólo cuando demuestran su habilidad de tener el efecto deseado en el mundo.

Las motivaciones son utilizadas en cada comportamiento motivacional, el cual es el mecanismo principal para alcanzar la selección adaptativa de acciones en esta arquitectura. En todo momento, durante la misión los comportamientos motivacionales reciben entradas de varias fuentes, incluyendo alimentación de los sensores, comunicación entre robots, alimentación de inhibición desde los otros comportamientos motivacionales, y motivaciones internas. Estas entradas se combinan para producir la salida del comportamiento motivacional.

Luego, esta salida define el nivel de activación del correspondiente conjunto de comportamientos, representado como un número no negativo. Cuando este nivel de activación excede cierto umbral, el conjunto de comportamientos asociado se activa.

Dos tipos de motivaciones son modeladas en ALLIANCE impaciencia y asentimiento. La motivación de impaciencia permite que un robot maneje situaciones donde otros robots fallan al momento de realizar cierta tarea. La motivación de asentimiento permite que un robot maneje situaciones en las cuales el mismo falla en realizar correctamente la tarea. Un comportamiento motivacional funciona de la siguiente manera. La motivación de un robot de activar algún conjunto de comportamientos es inicializada en 0. Luego con el tiempo, la motivación del robot por realizar un conjunto de comportamientos aumenta tan rápido como el parámetro de impaciencia, mientras esta tarea asociada a este grupo no sea realizada por ningún miembro del mismo. Si el robot r_i esta enterado de que el robot r_k esta trabajando en una tarea determinada, r_i estará satisfecho durante cierto período de tiempo que la tarea está siendo atendida sin su participación, y por esto ira en busca de otra tarea. La motivación de este robot para esa tarea continuará creciendo pero en menor medida.

Mientras que la característica de impaciencia refleja el hecho de que un robot pueda fallar, el asentimiento indica el reconocimiento que él mismo puede fallar. Esta característica funciona de la siguiente manera. Cuando un robot r_i está realizando una tarea, su voluntad de abandonarla aumenta con el tiempo mientras sus sensores demuestren que la tarea no está siendo completada. Tan pronto como otro robot r_k indique que comenzó la misma tarea que r_i y r_i se da cuenta de ello y que estuvo tratando durante un período de tiempo adecuado, el robot r_i deja esa tarea en un intento de encontrar una acción en la cual sea más productivo. Aún cuando la tarea no sea tomada por otro robot r_k, r_i dejara la tarea de todas formas si la tarea no es realizada en un tiempo razonable mayor al antes indicado. Esto permite que el robot r_i pase a realizar otra tarea más productiva en lugar de quedarse trancado en una tarea no productiva por siempre. Con esta característica de asentimiento, un robot está capacitado para adaptar sus acciones a fallas propias.

3.2.3. Formalización de la arquitectura

El problema se puede formalizar de la siguiente manera. Un conjunto de *n* robots heterogéneos que definen el equipo cooperativo $R=\{r_1, r_2, \ldots, r_n\}$. Un conjunto de *m* tareas que definen la misión $T=\{task_1, task_2, \ldots, task_m\}$. Cada robot en ALLIANCE dispone de conjuntos de comportamientos que determinan las capacidades del mismo, para referirnos a los conjuntos de comportamientos del robot r_i se define el conjunto $A_i=\{a_{i1}, a_{i2}, \ldots\}$. En ALLIANCE cada conjunto de comportamientos proporciona al robot de una función de realización de la tarea de alto nivel (high-level task-achieving), que se corresponden con las funciones brindadas en los niveles altos de la arquitectura de Brooks [Brooks1986]. Dado que los diferentes robots poseen diferentes conjuntos de comportamientos se necesita alguna manera de conocer que tarea realiza un robot cuando activa un determinado conjunto de comportamientos. Con este fin se definen *n* funciones de traducción $\{h_1(a_{1k}), h_2(a_{2k}), \ldots, h_n(a_{nk})\}$, donde $h_i(a_{ik}):A_i \rightarrow T$ retorna la tarea que realiza el robot r_i cuando activa su conjunto de comportamientos a_{ik}.

3.2.3.1 Umbral de activación

El umbral de activación de los conjuntos de comportamientos está dado por el parámetro θ. Este parámetro determina el valor de motivación que debe alcanzar un comportamiento motivacional para activar el conjunto de comportamientos controlado.

3.2.3.2 Información sensorial

La información sensorial le indica al comportamiento motivacional si corresponde activar el conjunto de comportamientos asociado a él. Esta señal contiene ruido y puede ser originada a partir de sensores reales o virtuales del robot. Se define la siguiente función que captura la noción de información sensorial para incluir en el cálculo de la motivación:

$$sensory\_feedback_{ij}(t) = \begin{cases} 1 & \text{if(la información sensorial en el robot } r_i \text{ en tiempo t} \\ & \text{indica que el conjunto de comportamiento } a_{ij} \text{ es aplicable)} \\ 0 & \text{en otro caso} \end{cases}.$$

En el diseño que se realizó como parte de este trabajo esta función se incluye dentro de cada comportamiento, pues cada comportamiento está capacitado por si mismo para decidir si es necesario activarse.

3.2.3.3 Comunicación entre robots

A continuación se describe el mecanismo de comunicación entre robots utilizado por ALLIANCE que le permite determinar que están haciendo los robots, además de permitirle saber cuando un robot se agregó al grupo o dejo de funcionar. El mecanismo de comunicación utiliza los parámetros ρ_i y τ_i, y la función *comm_received*. El primer parámetro indica cada cuanto tiempo informa un robot su actividad y el segundo indica el tiempo máximo que puede pasar sin recibir un mensaje de un robot antes de asumir que abandonó el grupo o ha dejado de funcionar. Por último, se define la función *comm_received* que permite conocer si un determinado robot está realizando una determinada tarea.

$$comm\_received(i,k,j,t_1,t_2) = \begin{cases} 1 & \text{if(el robot } r_i \text{ ha recibido un mensaje del robot } r_k \text{ referente} \\ & \text{a la tarea } h_i\ (a_{ij}) \text{ en el intervalo } (t_1, t_2)), \text{ donde } t_1 < t_2 \\ 0 & \text{en otro caso} \end{cases}$$

Como se verá más adelante si sólo se tiene en cuenta el intervalo (t_1,t_2), esta función es invocada por la arquitectura ALLIANCE de dos formas, haciendo referencia al intervalo (0,t) o (t,tiempoActual) por lo que permite implementarla conociendo únicamente el primero y el último de los mensajes para cada par (robot, conjunto de comportamientos) posible.

3.2.3.4 Supresión de activación de conjunto de comportamientos

Cada robot debe ejecutar sólo uno de sus comportamientos de alto nivel. La activación de los comportamientos de alto nivel es controlada por los comportamientos motivacionales, por lo tanto, cuando un comportamiento motivacional decide activar su conjunto de comportamientos debe simultáneamente evitar la activación de otros comportamientos de alto nivel. En la Figura 3 esta característica se muestra mediante las flechas de inhibición cruzada. Esta inhibición de otras actividades es modelada con la siguiente función:

$$activity\_suppression_{ij}(t) = \begin{cases} 0 & \text{if(otro conjunto de comportamiento } a_{ik} \text{ está activo, } k \neq j, \\ & \text{en el robot i en tiempo t)} \\ 1 & \text{en otro caso} \end{cases}.$$

Impaciencia del robot

Tres nuevos parámetros aparecen para implementar la característica de impaciencia propuesta en ALLIANCE: $\phi_{ij}(k,t)$, $\delta\_slow_{ij}(k,t)$ y $\delta\_fast_{ij}(t)$. El primer parámetro determina el tiempo durante el cual el robot r_i esta dispuesto a que un robot r_k trabaje en la tarea $h_i(a_{ij})$ sin impacientarse fuertemente con él. Los siguientes dos parámetros, $\delta\_slow_{ij}(k,t)$ y $\delta\_fast_{ij}(t)$, definen los parámetros de impaciencia para el robot r_i respecto al conjunto de comportamientos a_{ij} cuando hay otro robot realizando la tarea $h_i(a_{ij})$ o cuando no hay ningún robot realizándola respectivamente. En caso de que varios robots estén trabajando sobre la tarea $h_i(a_{ij})$ se utilizará el menor valor de δ_slow. Esto se formaliza en la siguiente función:

$$impatience_{ij}(t) = \begin{cases} \min_k \delta\_\text{slow}_{ij}(k,t) & if\ (comm\_received(i,k,j,t-\tau_i,t)=1) \\ & and\ (comm\_received(i,k,j,0,t-\phi_{ij}(k,t))=0) \\ \delta\_\text{fast}_{ij}(t) & \text{en otro caso} \end{cases}.$$

El detalle final es hacer que la motivación del conjunto de comportamientos a_{ij} sea llevada a cero la primera vez que el robot escucha que otro robot está realizando la tarea $h_i(a_{ij})$.

$$impatience\_reset_{ij}(t) = \begin{cases} 0 & if\ \exists k.(comm\_received(i,k,j,t-\delta t,t)=1) \\ & and\ (comm\_received(i,k,j,0,t-\delta t)=0), \\ & \text{where } \delta\text{t es el tiempo desde el último chequeo de comunicación} \\ 1 & \text{en otro caso} \end{cases}.$$

Asentimiento del robot

Dos parámetros son usados en ALLIANCE para modelar la característica de ceder a otro robot una tarea o darse por vencido (asentimiento): $\psi_{ij}(t)$ y $\lambda_{ij}(t)$. El primer parámetro, $\psi_{ij}(t)$, determina el tiempo que el robot r_i va a mantener activo el conjunto de comportamientos a_{ij} antes de cedérselo a otro robot. El segundo parámetro, $\lambda_{ij}(t)$, indica el tiempo que está dispuesto el robot r_i a mantener activo el conjunto de comportamientos a_{ij} antes de darse por vencido e intentar otra tarea. La siguiente función indica cuando un robot decide abandonar la tarea que estaba realizando:

$$acquiescence_{ij}(t) = \begin{cases} 0 & if\ [(\text{el conjunto de comportamiento } a_{ij} \text{ ha estado activo} \\ & \text{por mas de } \psi_{ij}(t) \text{ unidades de tiempo}) \\ & and\ (\exists x.comm\_received(i,x,j,t-\tau_i,t)=1)] \\ & or\ (\text{el conjunto de comportamiento } a_{ij} \text{ ha estado activo} \\ & \text{por mas de } \lambda_{ij}(t) \text{ unidades de tiempo}) \\ 1 & \text{en otro caso} \end{cases}.$$

Esta función retorna cero cuando determina que el robot debe ceder la tarea a otro miembro del equipo.

3.2.3.5 Cálculo de la motivación

A partir de las definiciones realizadas se puede indicar como se actualiza la motivación en cada uno de los comportamientos motivacionales, determinando el mecanismo central de selección de tareas. La motivación en el comportamiento motivacional a_{ij} está dada por la siguiente recurrencia:

$$
\begin{aligned}
& m(0)_{ij} = 0 \\
& m(t)_{ij} = \left[m(t-1)_{ij} + impatience_{ij}(t) \right] \times \\
& \qquad sensory\_feedback_{ij}(t) \times \\
& \qquad activity\_suppression_{ij}(t) \times \\
& \qquad impatience\_reset_{ij}(t) \times \\
& \qquad acquiescence_{ij}(t)
\end{aligned}
$$

En cada comportamiento motivacional la motivación por activar el conjunto de comportamientos asociado a él, crece según la impaciencia y es llevado a cero por el resto de las funciones que se incluyen en la definición.

3.2.4. Ambigüedades y contradicciones

En general, las ambigüedades observadas se deben a diferencias entre la descripción en lenguaje natural y la definición formal de la función. Parker intenta hacer una analogía entre los grupos de personas trabajando en equipo y su arquitectura con frases del estilo, “si determino que un robot está trabajando en una determinada tarea entonces me impacientaré lentamente respecto de esa tarea” y luego define la función *impatience* que solo se impacientará lentamente la primera vez que el robot realice esa tarea.

La primera contradicción observada en el modelo de ALLIANCE surge de la interpretación de la función *comm_received*, para la cual el texto que la describe dice así “each motivational behavior a_{ij} of robot r_i must also note when a team member is pursuing task $h_i(a_{ij})$” [Parker1998] y la definición formal de la función *comm_received*(i,k,j,t_1,t_2) indica que esta devuelve 1 si el robot r_i recibió un mensaje del robot r_k respecto a la tarea $h(a_{ij})$ en el intervalo de tiempo (t_1,t_2). ¿La función *comm_received* debe indicar si en algún momento el robot trabajó en la tarea o si el robot está trabajando en la tarea?

Si se trabaja con tareas que deben activarse varias veces durante una misma misión se debe tomar la definición que pregunta si el robot está trabajando en un determinado conjunto de comportamientos en un intervalo de tiempo dado. Esto permite que las dos razones de crecimiento (lento y rápido) tengan sentido. Esta interpretación a su vez entra en conflicto con la definición de *impatience_reset*, pues permite que la falla de un robot lleve a cero infinitas veces la impaciencia de otro comprometiendo de esta forma la finalización de la misión. Por ejemplo, un robot r_i que activa la tarea w, envía información de actividad, y debido a una falla la desactiva inmediatamente. Bajo esta suposición, todo robot que incluya esa tarea entre sus habilidades recibe el mensaje y lleva a cero la motivación para activar dicha tarea, esto se repite por siempre evitando que otros miembros del equipo ejecuten a la tarea w. En el siguiente capítulo se define un nuevo formalismo de control que resuelve este problema.

Respecto a la función *impatience_reset*, es necesario tener en cuenta que un robot recibe sus propios reportes de actividad y por lo tanto *k* debe ser distinto de *i*, luego la función debe ser redefinida de la siguiente manera:

$$impatience\_reset_{ij}(t) = \begin{cases} 0 & if \exists k.(k \neq i)\ and\ (comm\_received(i,k,j,t-\delta t,t)=1) \\ & and\ (comm\_received(i,k,j,0,t-\delta t)=0), \\ & \text{donde } \delta\text{t=tiempo desde el último chequeo de comunicación} \\ 1 & \text{en otro caso} \end{cases}.$$

En caso de que varios robots, decidan activar el conjunto de comportamientos asociado a una misma tarea simultáneamente o casi simultáneamente (entre su solape de δt), todos lo desactivarán inmediatamente de acuerdo a la función anterior, lo cual puede empeorar el rendimiento del equipo de robots. Esto se ve agravado por la interpretación dada a la función *comm_received.* Para resolverlo se debe volver a escribir la función de la siguiente manera:

$$impatience\_reset_{ij}(t) = \begin{cases} 0 & if \exists k.(k \neq i)\ and\ ((k<i) or ((k>i)\ and\ (m_{ij}(t-1)+impatience_{ij}(t)<\theta))) \\ & and\ (comm\_received(i,k,j,t-\delta t,t)=1) \\ & and\ (comm\_received(i,k,j,0,t-\delta t)=0), \\ & \text{donde } \delta\text{t=tiempo desde el último chequeo de comunicación} \\ 1 & \text{en otro caso} \end{cases}.$$

De esta forma, si varios robots activan simultáneamente una misma tarea, se prioriza la activación de los comportamientos de alto nivel para los robots con menor identificador.

En la definición de la función de asentimiento ocurre un problema similar al detectado en la función *impatience_reset* debido a la recepción de los propios reportes de actividad. Por lo que se debe ceder el trabajo a un robot r_k si éste lo intentó por más de ψ unidades de tiempo y *k* es distinto de *i.* Además, el hecho que un comportamiento esté activo por más de cierto tiempo se puede escribir formalmente utilizando la función *comm_received*. Por lo que la función debe ser redefinida de la siguiente manera:

$$acquiescence_{ij}(t) = \begin{cases} 0 & if\ [(comm\_received(i,i,j,0,t-\psi_{ij}(t))=1) \\ & and (\exists k.(k \neq i) and (comm\_received(i,k,j,t-\tau_i,t)=1))] \\ & or (comm\_received(i,i,j,0,t-\lambda_{ij}(t))=1) \\ 1 & \text{en otro caso} \end{cases}.$$

La definición de motivación no permite que dos o más robots trabajen sobre la misma tarea. Esto se ve claramente con el siguiente ejemplo. El robot r_k tiene activo el conjunto de comportamientos a_{kr}, luego otro robot r_i activa su conjunto de comportamientos a_{ij}, donde $h_i(a_{ij})=h_k(a_{kr})$. Según la definición de *impatience_reset* el robot r_k desactiva el conjunto de comportamientos. De esto se concluye que la inicialización de la impaciencia debe generarse cuando el robot no está trabajando en esa tarea, por lo que la nueva definición de la función es la siguiente:

$$impatience\_reset_{ij}(t) = \begin{cases} 0 & if\ \exists k.\left(k \neq i\right) and \left((k < i) or \left(\left(k > i\right) and \left(m_{ij}(t-1) + impatience_{ij}(t) < \theta\right)\right)\right) \\ & and \left(comm\_received\left(i,k,j,t-\delta t,t\right) = 1\right) \\ & and \left(comm\_received\left(i,k,j,0,t-\delta t\right) = 0\right) \\ & and \left(\prod_k activity\_suppression_{ij}(t) = 1\right), \\ & \text{where } \delta\text{t=tiempo desde el último chequeo de comunicación} \\ 1 & \text{en otro caso} \end{cases}.$$

Las modificaciones al formalismo que se proponen en esta sección achicar la brecha entre el formalismo y el diseño de la arquitectura, y de esta forma simplificar la implementación por parte del diseñador del equipo de robots. Si bien estos aportes son importantes los mismos oscurecen aún más a la arquitectura.

3.3. L-ALLIANCE

3.3.1. Introducción

Parker en su propuesta de ALLIANCE no define un mecanismo que adapte los parámetros de la arquitectura de acuerdo a cambios en el entorno o a la conformación del equipo, de manera de mejorar el rendimiento del equipo. Sumado a la gran cantidad de parámetros a definir puede tornarse duro utilizar ALLIANCE al momento de diseñar equipos de robots.

En L-ALLIANCE Parker define un mecanismo para ajustar los parámetros automáticamente con el objetivo de mejorar el rendimiento del equipo aún cuando ocurran cambios inesperados en el entorno o en el equipo. Esta nueva característica agregada a ALLIANCE tiene una serie de ventajas: no se requiere que un operador humano configure todos los parámetros de la arquitectura ALLIANCE, facilita el uso de sistemas multi-robots para cualquier tipo de aplicación, mejora el rendimiento del equipo, y permite la adaptación automática de los parámetros de modo de mejorar el rendimiento del equipo.

Como indica Parker, ALLIANCE no maneja varias cuestiones para obtener una arquitectura de control donde el grupo de robots controlados se desempeñe en forma eficiente. Estas son:

- Asegurar que los robots ejecuten las tareas para las cuales están mejor capacitados.
- Permitir que el equipo mejore el rendimiento con el tiempo.
- No permitir que una falla al realizar una tarea lleve a una falla total del robot.
- Minimizar el tiempo que los robots pasan ociosos.

Al igual que ALLIANCE, L-ALLIANCE define ciertas hipótesis de trabajo, éstas son:

- El promedio del rendimiento sobre los intentos recientes de un robot al realizar una determinada tarea es indicador razonable del rendimiento futuro del mismo.
- Si el robot r_i está monitoreando el entorno para determinar el rendimiento de otro robot r_k, y el entorno se ve modificado, estos cambios se atribuyen al robot.

En la Figura 5 se muestra la arquitectura propuesta para L-ALLIANCE Como se puede apreciar la única diferencia con la arquitectura ALLIANCE es la presencia de los monitores.

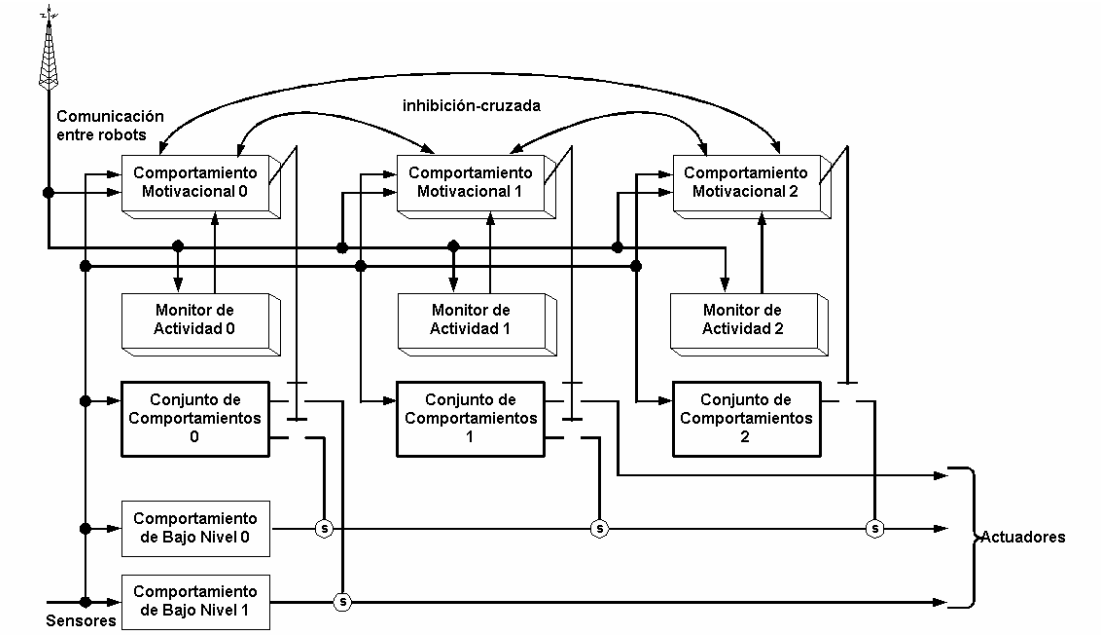


Figura 5. Diagrama de la arquitectura L-ALLIANCE.

3.3.1.1 Monitores de rendimiento

Cada monitor es responsable de observar y registrar el rendimiento de todo robot que participa de la misión, incluido él mismo, asociado al conjunto de comportamientos monitoreado. Formalmente el robot r_i es equipado con un conjunto de comportamientos de alto nivel $A_i=\{a_{i1}, a_{i2} ..., a_{ib}\}$ y ahora además con un conjunto de monitores $MON_i=\{mon_{i1}, mon_{i2} ..., mon_{ib}\}$, cada monitor, mon_{ij} observa el rendimiento de todo robot que realice la tarea $h_i(a_{ij})$ y registra el tiempo necesario para realizar dicha tarea.

3.3.1.2 Dos fases de control

En L-ALLIANCE, Parker distingue entre misiones de entrenamiento y misiones reales. El propósito de las misiones de entrenamiento es permitir que los robots se familiaricen con las capacidades y rendimientos propios y de sus compañeros de equipo sin preocuparse por terminar exitosamente la misión. Cuando se trata de una misión real el equipo debe asegurar el éxito de la misión de manera eficiente.

3.3.2. El problema de eficiencia

En el Capítulo 2 se mostró la definición reducida de Parker para el problema de la eficiencia para ALLIANCE (AEP), a continuación se muestra el problema tal cual fue propuesto en [Parker1998].

El AEP se define de la siguiente manera:

Dados

- $R = \{r_1, r_2, ..., r_r\}$ el conjunto de robots que conforman el equipo,
- A_i *conjunto de tareas que puede realizar el robot* r_i y
- $q : \cup_i A_i \rightarrow \Re^+$, $q(a_{ij})$ determina la calidad con que el robot r_i realiza la tarea $h_i(a_{ij})$.

Se desea determinar

- $U_i = \left\{ a_{ij} \middle| \textit{el robot } r_i \textit{ activa la tarea } \mathrm{h_i}(a_{ij}) \textit{ durante la misión} \right\}, \forall i \in \{0,..,n\}$.

Que minimice

- $\max_i \left(\sum_{a_{ij} \in U_i} q\left(a_{ij}\right) \right)$.

Sujeto a

- $A_i \subseteq T, \forall i \in \{1,...,n\}$ y
- $U_i \subseteq A_i, \forall i \in \{1,...,n\}$.

Al igual que los problemas de asignación presentados en el Capítulo 2, este problema de asignación es NP-difícil [Parker1998].

3.3.3. Mecanismos automáticos para la actualización de parámetros

L-ALLIANCE pretende mantener la tolerancia a fallas, principal virtud de su predecesora, y realizar una ejecución eficiente de la misión, lo cual se divide en los siguientes dos puntos:

1. Cómo determinan los robots individuales cuando interrumpir la tarea de un compañero, o cuando deben ceder una tarea a otro robot o para realizar otra tarea.
2. Cómo los robots seleccionan entre las tareas incompletas que no están siendo realizadas por ningún otro miembro del equipo.

En las siguientes secciones se presentan las diversas estrategias propuestas por Parker para abordar el problema de la eficiencia en ALLIANCE. Para atacar el punto uno, antes indicado, la autora de L-ALLIANCE define tres estrategias para la actualización de la impaciencia y el asentimiento. Por otro lado, para atacar el punto dos, ella define tres estrategias para ordenar las tareas.

3.3.3.1 Tres estrategias de actualización de la impaciencia y el asentimiento

La impaciencia y el asentimiento son las principales motivaciones introducidas en ALLIANCE, este mecanismo determina cuando ponerse impaciente con un robot y cuando ceder una tarea a otro robot o abandonarla para realizar otra tarea. Los parámetros que controlan este mecanismo, además de afectar la tolerancia a fallas afectan el rendimiento del equipo. Si los valores de asentimiento son pequeños el robot estará todo el tiempo cambiando de tarea y si los valores de asentimiento o impaciencia son muy altos un robot puede perder mucho tiempo ocioso o ejecutando una tarea para la cual ya no está capacitado.

Tabla 2. Valores de los parámetros para las estrategias de actualización motivacional.

Estrategia	Impaciencia ($\phi_{ij}(t)$)	Asentimiento($\psi_{ij}(t)$)
I	Tiempo propio	Tiempo propio
II	Tiempo propio	Tiempo mínimo del equipo
III	Tiempo del robot que ejecuta la tarea	Tiempo propio

Los parámetros principales que determinan la respuesta del robot r_i respecto del rendimiento del robot r_k en la tarea $h_i(a_{ij})$ en tiempo son: $\phi_{ij}(k,t)$, $\psi_{ij}(t)$ y $\lambda_{ij}(t)$, que determinan la impaciencia, el asentimiento a otro robot y el asentimiento a otra tarea respectivamente. En L-ALLIANCE se definen tres estrategias de control para la actualización automática del mecanismo de motivación, estas estrategias se resumen en la Tabla 2 y se describen a continuación.

Estrategia I: Desconfianza respecto al conocimiento del rendimiento de los compañeros

Esta estrategia necesita conocer únicamente la información de su propio rendimiento. Desconfía de la información que puede obtener de sus compañeros por lo que utiliza sólo su información de rendimiento para actualizar los parámetros de motivación.

En general, esta estrategia es utilizada cuando se tiene poca información del equipo, por ejemplo la primera vez que un grupo de robots trabajan juntos, o cuando se tiene poca confianza de la información recabada de los compañeros de equipo.

Cuando un robot utiliza esta estrategia y sabe que necesita cierto tiempo para completarla se pondrá impaciente con todo robot que trabaja en dicha tarea durante un tiempo mayor al tiempo requerido por él, y estará dispuesto a ceder una tarea si ha trabajado en ella más de dicho tiempo.

Estrategia II: Dejar que el mejor robot gane

Bajo esta estrategia un robot que esté realizando una determinada tarea la cederá, si hay otro robot tratando de ejecutarla y él mismo estuvo trabajando en la misma más del tiempo requerido por el robot que mejor sabe ejecutarla, y se pondrá impaciente con los robots que demoran más del tiempo requerido por él para ejecutar la tarea.

Estrategia III: Dar una oportunidad de trabajar

En esta estrategia el robot juzga el rendimiento de sus compañeros basándose en el conocimiento que él mismo tiene de sus compañeros, en lugar de compararlos con los robots que mejor saben realizar la tarea. De esta forma, no se pondrá impaciente con los robots mientras estos se comporten según la información de rendimiento recabada y cederá una tarea cuando el tiempo de ejecución supere el tiempo medio requerido normalmente por él.

3.3.3.2 Tres estrategias para ordenar tareas

El segundo punto a tener en cuenta para mejorar el rendimiento es determinar en que orden eligen los robots la siguiente tarea a realizar entre las tareas que no están siendo realizadas por otro miembro del equipo (tareas libres). En ALLIANCE, la elección de la siguiente tarea libre a ejecutar está determina por el parámetro δ_fast. Las tareas con mayor valor de δ_fast serán elegidas primero. Al modificar este parámetro se obtienen distintas estrategias de selección de tareas libres. Se presentan a continuación las tres estrategias estudiadas en L-ALLIANCE: la tarea más larga primero, la tarea más corta primero y selección aleatoria de tareas.

La tarea más larga primero

Cada robot elige la tarea libre más larga para ejecutar. Esto se logra ajustando el parámetro δ_fast proporcional a la duración esperada de la tarea por el robot. La idea tras esta estrategia es ejecutar la tarea más larga en paralelo con varias tareas pequeñas. Esta estrategia tiene un bajo desempeño cuando los robots son heterogéneos pues los robots que peor saben realizar una tarea serán asignados a ella.

Pseudo tarea más corta primero

Luego de definir la estrategia que elige la tarea más larga primero resulta intuitivo definir la estrategia que elige la tarea más corta. Parker, en lugar de esto, define esta estrategia en la cual todo robot clasifica primero a las tareas en dos categorías:

1. Las tareas que el robot sabe realizar mejor que cualquier otro.
2. El resto de las tareas.

Luego cada robot elige primero la tarea más corta dentro de la categoría 1 y luego considera las tareas de la categoría 2.

Para elegir la tarea más corta dentro de la categoría 1 es necesario ajustar el parámetro δ_fast inversamente proporcional a la duración esperada de la tarea por el robot.

Pseudo selección aleatoria de tarea

En esta estrategia se dividen las tareas en las dos categorías definidas en la estrategia anterior. Luego los robots eligen tareas al azar primero en la categoría uno y luego en la dos.

3.3.3.3 Mecanismo automático preferido

Parker realiza una serie de experimentos simulando diversas condiciones del entorno y del equipo, tratando de encontrar una estrategia automática que se comporte adecuadamente.

Para esto introduce el concepto de progreso cuando se trabaja (Progress When Working) (PWW), el cual, en caso de cumplirse determina que cuando un robot trabaja en una determinada tarea decrementa el tiempo necesario para finalizarla. En el Capítulo 4 se muestra la definición formal de PWW.

Luego ejecuta varias misiones simuladas variando la cantidad de tareas, la cantidad de robots, el grado de heterogeneidad entre los robot y el indicador PWW.

Al analizar los datos generados en la simulación, la estrategia preferida por Parker para la actualización de la impaciencia y el asentimiento para equipos homogéneos donde no se cumpla la condición PWW es la III, en otro caso prefiere la II.

La estrategia preferida por Parker para ordenar tareas utiliza el siguiente algoritmo:

1. Cada robot divide las tareas en dos categorías:
    a. Las tareas que él cree realizar mejor que el resto del equipo y ningún otro miembro está realizando
    b. El resto de las tareas.
2. Elegir primero dentro de las tareas de la categoría (a) según la tarea más larga. En caso que la categoría (a) se vacía elegir de la categoría (b) según la tarea más corta.

3.3.4. Formalización de la arquitectura

En la sección anterior se presentó la estrategia preferida para L-ALLIANCE. En esta sección se explica como implementar esta estrategia sobre la formalización de ALLIANCE. Es importante aclarar que se utiliza la notación introducida en ALLIANCE tanto para los parámetros como para las funciones.

3.3.4.1 Definiciones preliminares

A continuación se definen varias funciones que son usadas para definir el modelo formal de L-ALLIANCE.

La función *robot_present*(i,t) devuelve el conjunto de robots que para el robot r_i están presentes y asignados a la misión. Esto es,

$$robot\_present(i,t) = \left\{ k \,\middle|\, \exists j.\left(comm\_received(i,k,j,t-\tau_i,t) = 1 \right) \right\}.$$

La función $learning\_impatience_{ij}(t)$ retorna 0 cuando algún robot, de los actualmente presentes en el equipo, trabajó en la tarea $h_i(a_{ij})$ y 1 en otro caso.

$$learning\_impatience_{ij}(t) = \begin{cases} 0 & if \left(\sum_{x \in robots\_present(i,t)} comm\_received(i,x,j,0,t) \right) \neq 0 \\ 1 & \text{en otro caso} \end{cases}.$$

Uno de los agregados a la arquitectura de ALLIANCE son los monitores de rendimiento, para acceder a la información recabada por estos se define la siguiente función:

$$task\_time_i(k,j,t) = \text{el promedio de tiempo, medido por el robot } r_i, \text{ de los } \mu \text{ pasados intentos del robot } r_k \text{ al realizar la tarea } h_i(a_{ij}) \text{ adicionando la desviación estándar}.$$

La estrategia preferida para ordenar tareas define dos categorías de tareas, para formalizar está división se define la siguiente función:

$$task\_category_{ij}(t) = \begin{cases} 1 & if\left(task\_time_i(i,j,t) = \min_{k \in robots\_presents(i,t)} task\_time_i(k,j,t)\right) \\ & and\left(\left(\sum_{k \in robots\_presents(i,t)} comm\_received(i,k,j,t-\tau_i,t)\right) = 0\right) \\ 2 & \text{en otro caso} \end{cases}.$$

La siguiente función define una nueva motivación que determina el grado de aburrimiento de un robot. Inicialmente el aburrimiento (boredom) de un robot es cero. Con el tiempo, este aumenta según un nuevo parámetro denominado boredom_rate y es llevado a cero toda vez que el robot trabaja en una tarea.

$$boredom_i(t) = \begin{cases} 0 & if(t=0) \\ [boredom_i(t-1) + boredom\_rate_i] \times \prod_j activity\_suppression_{ij}(t) & \text{en otro caso} \end{cases}.$$

La función *learned_robot_influence* es utilizada para controlar el crecimiento de la motivación. La misma permite crecer la motivación de las tareas en la categoría 1 mientras tranca el crecimiento de las tareas en la categoría 2 cuando el robot no está aburrido. Se definen que un robot está aburrido cuando el aburrimiento es mayor o igual a un determinado umbral de aburrimiento (*boredom_threshold*). La misma se define de la siguiente manera:

$$learned\_robot\_influence_{ij}(t) = \begin{cases} 0 & if\left(boredom_i(t) < boredom\_threshold_i\right) \\ & and\left(task\_category_{ij}(t) = 2\right) \\ 1 & \text{en otro caso} \end{cases}.$$

3.3.4.2 Implementando la actualización de la impaciencia y el asentimiento

La estrategia preferida determina que cuando no se cumple la condición PWW y el equipo es homogéneo[7] debe regir la estrategia III y en otro caso la estrategia II. Para regular la impaciencia y el asentimiento se debe indicar como ajustar los parámetros ϕ, ψ, y δ_slow de forma de implementar las estrategias II y III.

Cuando se quiere implementar la estrategia II el parámetro ϕ debe tener el valor del tiempo propio y en caso de querer implementar la estrategia III debe tener el valor del tiempo requerido por el robot que está realizando la tarea. A continuación se formaliza la asignación del parámetro.

$$\phi_{ij}(k,t) = \begin{cases} taskTime_i(k,j,t) & if(\neg PWW) and (heterogeneity = MILD) \\ taskTime_i(i,j,t) & \text{en otro caso} \end{cases}.$$

[7] Parker define un equipo heterogéneo MILD cuando el grado de heterogeneidad es menor a 300%.

De igual forma, cuando se quiere implementar la estrategia II el parámetro ψ debe tener el valor del tiempo requerido por el robot que mejor sabe hacer la tarea y en caso de querer implementar la estrategia III debe tener el valor del tiempo propio. A continuación se formaliza la asignación del parámetro.

$$\psi_{ij}(t) = \begin{cases} taskTime_i(i,j,t) & if\ (\neg PWW)\ and\ (heterogeneity = MILD) \\ \min\limits_{k \in robots\_present(i,t)} taskTime_i(k,j,t) & \text{en otro caso} \end{cases}.$$

El último parámetro requerido para la actualización de la impaciencia de forma que se respeten las estrategias es δ_slow. Este parámetro debe ajustarse para permitir que los robots ejecuten sus tareas sin que otro robot se ponga impaciente con él siempre y cuando se respeten el tiempo asignado por la estrategia. El tiempo asignado por la estrategia según las condiciones del entorno y del equipo está dado por ϕ. Por lo tanto, si se ajusta δ_slow de acuerdo a la siguiente expresión se respetaran los tiempos determinados por la estrategia

$$\delta\_slow_{ij}(k,t) = \theta / \phi_{ij}(k,t)\ .$$

3.3.4.3 Implementando estrategias para ordenar tareas

La forma de implementar el mecanismo de elección entre las tareas que no son ejecutadas por ningún robot se relaciona directamente con el valor δ_fast. La tarea con mayor δ_fast es asignada primero. Se definen los parámetros *min_delay* y *max_delay* que indican el tiempo mínimo y máximo que puede pasar un robot ocioso, respectivamente. Luego para realizar el ajuste de δ_fast se sigue:

$$high = \max_{kj} task\_time_i(k,j,t)$$

$$low = \min_{kj} task\_time_i(k,j,t)$$

$$scale\_factor = \frac{max\_delay - min\_delay}{high - low}$$

$$\delta\_fast_{ij}(t) = \begin{cases} \dfrac{\theta}{min\_delay + (task\_time_i(i,j,t) - low) * scale\_factor} & if\ (task\_category_{ij}(t) = 2 \\ \dfrac{\theta}{max\_delay - (task\_time_i(i,j,t) - low) * scale\_factor} & en\ otro\ caso \end{cases}.$$

Esta forma de adaptar δ_fast asegura que los robots no estarán ociosos por más de *max_delay* unidades de tiempo. Además para las tareas de categorías 2 asigna mayores valores a las tareas más cortas y para las tareas de categoría 1 asigna mayores valores a las tareas más largas. Lo cual, como se verá más adelante, combinado con la función *learned_robot_influence* logra forzar la arquitectura para implementar la estrategia preferida.

3.3.4.4 Cálculo de la motivación

Para el cálculo de la motivación L-ALLIANCE distingue entre las distintas etapas.

Cuando el equipo se encuentra en la etapa de aprendizaje activo utiliza el siguiente mecanismo de actualización:

$$
\begin{aligned}
&m(0)_{ij} = 0 \\
&m(t)_{ij} = \Big[m(t-1)_{ij} + random\_increment \Big] \times \\
&sensory\_feedback_{ij}(t) \times \\
&activity\_suppression_{ij}(t) \times \\
&learning\_impatience_{ij}(t)
\end{aligned}
\quad .
$$

where

$$random\_increment = \theta \times (\text{un número al azar entre 0 y 1})$$

De esta forma permite que los robots conozcan tanto sus capacidades como las de sus compañeros.

Cuando el equipo se encuentra en la etapa adaptativa utiliza el siguiente mecanismo de actualización:

$$
\begin{aligned}
&m(0)_{ij} = 0 \\
&m(t)_{ij} = \Big[m(t-1)_{ij} + impatience_{ij}(t) \Big] \times \\
&\quad sensory\_feedback_{ij}(t) \times \\
&\quad activity\_suppression_{ij}(t) \times \\
&\quad impatience\_reset_{ij}(t) \times \\
&\quad acquiescence_{ij}(t) \times \\
&\quad learned\_robot\_influence_{ij}(t)
\end{aligned}
\quad .
$$

Este mecanismo es similar al propuesto en ALLIANCE si se elimina la función *learned_robot_influence*, la cual permite que los robots consideren primero a las tareas en la categoría 1.

3.3.5. Ejemplos de asignación no-óptima

Se han encontrado varias situaciones para las cuales L-ALLIANCE no asigna de forma óptima. La siguiente tabla muestra los tiempos que demoran en ejecutar dos robots determinadas tareas.

Tabla 3. Ejemplo 1 - Asignación no óptima

Tarea	**Robot r_0**	**Robot r_1**
task$_0$	2	3
task$_1$	1	4

Para estos valores L-ALLIANCE asigna al r_0 la tarea $task_0$ y al r_1 la tarea $task_1$. El tiempo total de la misión será entonces de 4 unidades cuando en realidad la solución óptima debería haber asignado al r_0 la tarea $task_1$ y al r_1 la tarea $task_0$, logrando de esta forma un tiempo de 3 unidades.

En este caso se logra la asignación óptima si se utiliza la tarea más corta como estrategia de ordenamiento de tareas.

Tabla 4. Ejemplo 2 - Asignación no óptima

Tarea	Robot r_0	Robot r_1
$task_0$	9	10
$task_1$	8	-

Para los valores expresados en la Tabla 4, L-ALLIANCE asigna al r_0 la tarea $task_0$ y r_1 queda ocioso. El tiempo total de la misión será entonces de 17 unidades cuando en realidad la solución óptima debería haber asignado al r_0 la tarea $task_1$ y al r_1 la tarea $task_0$, logrando de esta forma un tiempo de 10 unidades.

En este caso, nuevamente, logra la asignación óptima si se utiliza la tarea más corta como estrategia de ordenamiento de tareas. Pero, como se muestra en el siguiente ejemplo, esto sólo no es suficiente.

Tabla 5. Ejemplo 3 - Asignación no óptima

Tarea	Robot r_0	Robot r_1
$task_0$	8	10
$task_1$	10	-

Para los valores expresados en la Tabla 5, L-ALLIANCE asigna al r_0 la tarea $task_0$ y r_1 queda ocioso. El tiempo total de la misión será entonces de 18 unidades cuando en realidad la solución óptima debería haber asignado al r_0 la tarea $task_1$ y al r_1 la tarea $task_0$, logrando de esta forma un tiempo de 10 unidades.

En este caso logra la asignación óptima si se utiliza la tarea más larga como estrategia de ordenamiento de tareas.

En los dos últimos ejemplos se logra asignar de forma óptima si se asignan primero las tareas para las cuales hay menor cantidad de robots capaces de realizarlas. Esta idea es utilizada en el Capítulo 5 para definir nuevas estrategias de elección de tareas.

3.3.6. Algunas consideraciones

3.3.6.1 Respecto de las dos etapas de aprendizaje

L-ALLIANCE incluye una etapa de aprendizaje activo que permite a los robots conocer el desempeño de todos los miembros del equipo para realizar las distintas tareas. Luego, la información recabada por los monitores de actividad en la etapa de aprendizaje activo es utilizada para conocer las características de los robots que conforman el equipo. Por último, esta información es utilizada en la etapa de aprendizaje adaptativo para realizar el ajuste automático de parámetros.

Una de las características a resaltar de ALLIANCE es permitir el ingreso de nuevos miembros al equipo durante la misión. L-ALLIANCE no define el mecanismo de descubrimiento de nuevas capacidades o de nuevos miembros al equipo.

En la implementación de L-ALLIANCE que se realizó toda vez que un robot gana una nueva capacidad o se agrega un nuevo miembro, se realiza un intercambio de rendimientos entre todos los miembros del equipo que comparten dicha tarea. Esta información es utilizada para inicializar la información de rendimientos una única vez, utilizando luego la información recabada por los monitores de actividad durante la ejecución de la misión. En caso de que un robot no reciba información de rendimiento se asigna el tiempo propio. Esto elimina la necesidad de disponer de la etapa de aprendizaje activo.

3.3.6.2 Respecto de los parámetros y al formalismo

L-ALLIANCE es concebida para automatizar la configuración de los parámetros y la adaptación de los mismos a diversos cambios. Pero, en el lugar de remitirse únicamente a los parámetros, L-ALLIANCE modifica la definición funcional de ALLIANCE agregando, como se indico antes, fases de aprendizaje y diversas funciones. Al realizar estas modificaciones deberían verificarse las propiedades o bondades de ALLIANCE, en particular la prueba de terminación demostrada en [Parker1998], pues al modificar el formalismo no se garantiza que dichas propiedades se conserven.

Sería interesante mantener el formalismo de ALLIANCE y modificar únicamente los parámetros. En particular, podría eliminarse la función *learned_robot_influence* junto con el resto de funciones y parámetros que la componen, y concentrarse en ajustar los parámetro de ALLIANCE según que estrategia desea implementar.

3.3.6.3 Hacia un núcleo de ALLIANCE

Pensando en no complicar la arquitectura e incluso hacerla más simple, definiendo una especie de núcleo, aparecen las siguientes preguntas:

- ¿Deben existir dos parámetros de impaciencia o debe ser sólo uno ajustándose adecuadamente a las condiciones del problema?
- ¿Deben existir dos parámetros para determinar la motivación de asentimiento?
- ¿Debe un robot impacientarse lentamente cuando otro robot está trabajando en una tarea?
- ¿Debe un robot llevar a cero su impaciencia por algún motivo (*impacience_reset*)?

En general, Parker propone modificaciones a la arquitectura ALLIANCE para cada problema que debe resolver. Esto se puede verificar en [Parker1999] y en [Parker1998] donde modifica la definición de ALLIANCE.

Los puntos mencionados en esta sección dan lugar al desarrollo de un núcleo para ALLIANCE el cual se define en el Capítulo 4.

4. N-ALLIANCE

4.1. Introducción

Luego del análisis y la implementación de ALLIANCE y L-ALLIANCE se observaron detalles de su modelo formal que contradecían las intenciones de su diseñador o restringían demasiado las posibilidades al momento de diseñar un equipo cooperativo.

Se observó que L-ALLIANCE se apoya parcialmente en su predecesora, en particular modifica la actualización de la motivación y agrega las dos fases o etapas de control, complicando innecesariamente la arquitectura y perdiendo todas las demostraciones formales realizadas sobre ALLIANCE. Es una práctica habitual de Parker modificar el formalismo de ALLIANCE, adaptándolo a las necesidades de cada problema, en lugar de aprovechar la infraestructura brindada por ALLIANCE para resolver el problema planteado.

Por estas razones se propone un núcleo de control tolerante a fallas tomando las principales bondades de ALLIANCE y se extiende de forma que permita diseñar equipos de robots cooperativos más flexibles y más simples.

4.2. N-ALLIANCE: un núcleo para ALLIANCE

La motivación que impulso la definición de este núcleo radica principalmente en ciertas restricciones que obliga a seguir el modelo formal de ALLIANCE.

El modelo formal de ALLIANCE toma muy en cuenta el hecho de que un robot este trabajando en una determinada tarea. Tanto el valor de impaciencia como el de asentimiento se ve afectado si existe o no un robot realizando la tarea. Esto lleva a plantearse las siguientes preguntas. ¿Por qué debe un robot impacientarse menos respecto de una tarea cuando existen robots trabajando en ella? ¿Por qué puede un robot tomarse más tiempo para trabajar cuando está realizando la tarea solo? ¿Por qué debe un robot llevar a cero su motivación la primera vez que un compañero realiza la tarea? ¿Por qué debe un robot impacientarse lentamente la primera vez que un compañero activa una tarea y en posteriores activaciones no?

Uno de los objetivos de L-ALLIANCE es realizar el ajuste automático de los parámetros necesarios para definir el comportamiento cooperativo en ALLIANCE, pero mientras resuelve el ajuste de los mismos introduce una serie de parámetros, aunque más pequeña si la comparamos con ALLIANCE, que deben ser ajustados por el diseñador del equipo.

Además de lo expuesto antes, al momento de definir L-ALLIANCE, Parker redefine la actualización de la motivación e introduce las dos etapas de control. Es más adecuado proponer el mecanismo automático de actualización (L-ALLIANCE) utilizando ALLIANCE tal cual fue definida, en lugar de definir una nueva arquitectura muy similar a ALLIANCE para resolver el nuevo desafío. Esta propuesta conserva los resultados probados sobre ALLIANCE, por lo que no obliga a probar nuevamente las bondades para L-ALLIANCE. En particular, Parker demuestra la finalización de la misión asignada a un equipo que utiliza ALLIANCE como arquitectura de control [Parker1998], esta propiedad debería demostrarse nuevamente para L-ALLIANCE pues sus formalismos difieren.

En la definición de N-ALLIANCE no importa si un robot está realizando o no la tarea. Este se rige por el valor actual de impaciencia y asentimiento independientemente de la cantidad de robots que trabajen en la misma.

4.2.1. Formalización de N-ALLIANCE

El formalismo de la arquitectura de N-ALLIANCE difiere del definido en ALLIANCE pero el esquema de su arquitectura es el mismo (Figura 4).

N-ALLIANCE toma las ideas centrales de ALLIANCE y las utiliza para definir un mecanismo de control más simple y flexible. La simplificación introducida redunda en una cantidad menor de parámetros a configurar.

N-ALLIANCE considera a un conjunto de *n* robots heterogéneos que definen el equipo cooperativo $R=\{r_1, r_2, \ldots, r_n\}$, un conjunto de *m* tareas que definen la misión $T=\{task_1, task_2, \ldots, task_m\}$, un conjuntos de comportamientos $A_i=\{a_{i1}, a_{i2}, \ldots\}$ para cada robot r_i y una familia de funciones $\{h_1(a_{1k}), h_2(a_{2k}), \ldots, h_n(a_{nk})\}$ tal que cada elemento $h_i(a_{ik})$ permite conocer cual es la tarea (perteneciente a T) que está llevando a cabo el robot r_i usando su comportamiento de alto nivel a_{ik}.

4.2.1.1 Umbral de activación

El umbral de activación de los conjuntos de comportamientos está dado por el parámetro θ. Este parámetro determina el valor de motivación que debe alcanzar un comportamiento motivacional para activar el conjunto de comportamientos controlado por él.

4.2.1.2 Información sensorial

La información sensorial le indica al comportamiento motivacional si corresponde activar el conjunto de comportamientos asociado a él. Esta señal contiene ruido y puede ser originada a partir de sensores reales o virtuales del robot. Se define la siguiente función que captura la noción de información sensorial para incluir en el cálculo de la motivación:

$$sensoryFeedback_{ij}(t) = \begin{cases} 1 & if\,(si\ el\ sistema\ sensorial\ del\ robot\ r_i\ en\ tiempo\ t\ indica \\ & que\ el\ conjunto\ de\ comportamiento\ a_{ij}\ es\ aplicable) \\ 0 & en\ otro\ caso \end{cases}$$

.

Esta definición es idéntica a la utilizada en ALLIANCE.

4.2.1.3 Finalización de la tarea

Esta función no existe en ALLIANCE y se utiliza en N-ALLIANCE para indicar que el robot terminó de realizar la tarea asignada. Al igual que la función anterior *end* se asocia al sistema sensorial del robot. La función que formaliza este concepto es la siguiente:

$$end_{ij}(t) = \begin{cases} 0 & if\,(el\ sistema\ sensorial\ del\ robot\ r_i\ en\ tiempo\ t\ indica \\ & que\ el\ conjunto\ de\ comportamiento\ a_{ij}\ finalizó) \\ 1 & en\ otro\ caso \end{cases}$$

.

La definición de esta función no fue tenida en cuenta en ALLIANCE, ya que esta no formaliza el hecho que un robot termina su tarea debe iniciar a cero la motivación, permitiendo de esta forma elegir una nueva tarea a realizar.

4.2.1.4 Comunicación entre robots

Al igual que en ALLIANCE se describe el uso de la comunicación entre robots que le permite a N-ALLIANCE determinar que están haciendo los robots, además de permitirle saber cuando un robot se agregó al grupo o dejo de funcionar. Para esto utiliza los parámetros ρ_i y τ_i, y la función *commReceived*. El primer parámetro indica cada cuanto tiempo informa un robot su actividad y el segundo indica el tiempo máximo que puede pasar sin recibirse un mensaje de un robot antes de asumir que abandonó el grupo o ha dejado de funcionar. Por último, se define la función *commReceived* que permite consultar el intercambio de mensajes entre robots.

$$commReceived(i,k,j,t_1,t_2) = \begin{cases} 1 & if\ (el\ robot\ r_i\ ha\ recibido\ un\ mensaje\ del\ robot\ r_k\ referente \\ & a\ la\ tarea\ h_i(a_{ij})\ en\ el\ intervalo\ (t_1,t_2)),\ donde\ t_1 < t_2 \\ 0 & en\ otro\ caso \end{cases}.$$

Esta función se utiliza como base para determinar en que tarea está trabajando un robot, cuanto tiempo hace que está trabajando y cuantos robots funcionando hay en el equipo.

4.2.1.5 Supresión de activación de conjunto de comportamientos

En un momento dado, cada robot puede ejecutar sólo uno de sus comportamientos de alto nivel. Dado la activación de los comportamientos de alto nivel es controlada por los comportamientos motivacionales, cuando un comportamiento motivacional decide activar su conjunto de comportamientos debe simultáneamente evitar la activación de otros comportamientos de alto nivel. Esta inhibición de otras actividades es modelada con la siguiente función:

$$activitySuppression_{ij}(t) = \begin{cases} 0 & if\ (otro\ conjunto\ de\ comportamiento\ a_{ik}\ está\ activo,\ k \neq j, \\ & en\ el\ robot\ i\ en\ tiempo\ t) \\ 1 & en\ otro\ caso \end{cases}.$$

Más adelante se formalizarán otros conceptos que permiten dar una definición más formal de esta función.

4.2.1.6 Impaciencia del robot

En la definición de esta motivación es donde aparece la primera gran simplificación respecto de la arquitectura ALLIANCE. ALLIANCE utilizaba los parámetros $\phi_{ij}(k,t)$, $\delta\_slow_{ij}(k,t)$ y $\delta\_fast_{ij}(t)$ para definir esta función, la nueva definición de impaciencia utiliza sólo un parámetro, $\delta_{ij}(t)$, el cual indica el valor con el cual el robot r_i se impacienta respecto de la tarea $h_i(a_{ij})$.

En este punto también se elimina la función *impatience_reset*, la cual no cumple con su cometido cuando se trata de tareas repetitivas.

4.2.1.7 Asentimiento del robot

Dos parámetros son usados en ALLIANCE para modelar la característica de ceder a otro robot una tarea o darse por vencido (asentimiento): $\psi_{ij}(t)$ y $\lambda_{ij}(t)$. El asentimiento de un robot se ha simplificado con la eliminación del parámetro $\psi_{ij}(t)$. El parámetro mantenido, $\lambda_{ij}(t)$, indica el tiempo que está dispuesto el robot r_i a mantener activo el conjunto de comportamientos a_{ij} antes de darse por vencido e intentar otra tarea.

La siguiente función indica cuando un robot decide abandonar la tarea que estaba realizando:

$$acquiescence_{ij}(t) = \begin{cases} 0 & if\,(el\ conjunto\ de\ comportamiento\ a_{ij}\ ha\ estado\ activo \\ & \quad por\ más\ de\ \lambda_{ij}(t)\ unidades\ de\ tiempo) \\ 1 & en\ otro\ caso \end{cases}.$$

4.2.1.8 Cálculo de la motivación

Luego de está formalización se puede indicar como se actualiza la motivación en cada uno de los comportamientos motivacionales. La motivación en el comportamiento motivacional a_{ij} está dada por la siguiente recurrencia:

$$\begin{aligned} motivation(0)_{ij} = \;& 0 \\ motivation(t)_{ij} = \;& \left[motivation_{ij}(t-1) + \delta_{ij}(t)\right] \times \\ & sensoryFeedback_{ij}(t) \times \\ & activitySuppression_{ij}(t) \times \\ & acquiescence_{ij}(t) \times \\ & end_{ij}(t) \end{aligned}.$$

Se puede apreciar que en esta nueva definición se sustituye la función de *impatience* por δ, se elimina la función *impatience_reset* y se agrega la función *end*.

4.2.1.9 Otros formalismos

Se puede formalizar la idea de comportamiento de alto nivel activo con la función:

$$active_i(t) = \begin{cases} j & \\ & if\ \ \exists j.\left[\begin{array}{l} \left(motivation_{ij}(t) \geq \theta\right) \wedge \\ \left(\forall k.\left(k \neq j\right)\left(motivation_{ij}(t) > motivation_{ik}(t)\right) \vee\right) \\ \left(\left(motivation_{ij}(t) = motivation_{ik}(t)\right) \wedge \left(j < k\right)\right) \end{array}\right] \\ 0 & en\ otro\ caso \end{cases}.$$

La función active define cuando un comportamiento de alto nivel está activo y define como resolver conflictos entre comportamientos de alto nivel, cuando presentan niveles de motivación superiores al umbral de motivación.

Se define tiempo de ejecución de una tarea con la función:

$$workingTime_i(j,t) = \begin{cases} 0 & if\,(active_i(j) = 0) \\ time & en\ otro\ caso \end{cases}.$$

where time es el tiempo transcurrido desde que el robot r_i activó a_{ij}

A su vez, como se muestra a continuación, a partir estas definiciones se puede dar una definición más formal a las funciones *activitySuppression* y *acquiescence*.

$$activitySuppression_{ij}(t) = \begin{cases} 0 & if\,(active_i(t) \neq j) \\ 1 & en\ otro\ caso \end{cases}$$

$$acquiescence_{ij}(t) = \begin{cases} 0 & if\,(workingTime_i(j,t) > \lambda_{ij}(t)) \\ 1 & en\ otro\ caso \end{cases}$$

La función *robotsPresents* fue introducida en L-ALLIANCE, para formalizar dicha arquitectura. Como los comportamientos motivacionales internamente conocen la formación del equipo no es necesario invertir cálculo extra por lo que se incluye dentro de la definición de N-ALLIANCE y se define de la siguiente manera:

$$robotsPresents(i,t) = \left\{ k \,\middle|\, \exists j.\left(commReceived(i,k,j,t-\tau_i,t) = 1 \right) \right\}.$$

4.2.2. Implementación de ALLIANCE sobre N-ALLIANCE

N-ALLIANCE se desarrolló con el objetivo de brindar mayor flexibilidad al diseñador de equipos robóticos cooperativos y al mismo tiempo conservar todas las características sobresalientes de ALLIANCE.

N-ALLIANCE simplifica el formalismo propuesto en ALLIANCE pero permite construir sobre ella una arquitectura similar a ALLIANCE. La eliminación del mecanismo de inicialización de la motivación (*impatience_reset)* no permite reproducir exactamente a la arquitectura ALLIANCE sobre N-ALLIANCE. El mecanismo de inicialización lleva a cero la motivación de el comportamiento motivacional a_{ij} la primera vez que el robot r_i determina que otro robot está realizando la tarea $h_i(a_{ij})$. Este mecanismo fue eliminado pues cuando se trabaja con tareas repetitivas llevar a cero sólo una vez no tiene el efecto deseado, pues sólo llevará a cero los comportamientos motivacionales la primera vez que se ejecute. Por otro lado, si se lleva a cero toda vez que un robot comienza a realizar la tarea, un robot defectuoso o mal intencionado puede comprometer la finalización de la misión. Esto último ocurre, cuando un robot defectuoso activa el comportamiento motivacional asociado a la tarea *w*, e inmediatamente debido a su mal funcionamiento lo desactiva, llevando a cero todas las motivaciones asociados a la tarea w de los robot que saben realizarla, eliminando la posibilidad de que otro robot intente ejecutar la tarea.

En N-ALLIANCE los valores de impaciencia y asentimiento para una tarea no toman en cuenta el hecho de que haya o no un robot trabajando en ella, mientras que ALLIANCE define valores diferentes de impaciencia y asentimiento para los casos en que hay robots trabajando y los casos en que no. Por este motivo al construir ALLIANCE sobre N-ALLIANCE es necesario mantener una mayor cantidad de parámetros y ajustar adecuadamente los parámetros de N-ALLIANCE en función de si hay o no robots trabajando.

Esta implementación debe mantener los parámetros $\psi_{ij}(t)$, $\lambda_{ij}(t)$, $\phi_{ij}(k,t)$, $\delta\_slow_{ij}(k,t)$ y $\delta\_fast_{ij}(t)$, utilizados para ajustar la motivación en ALLIANCE. El parámetro $\lambda_{ij}(t)$ se renombra a $\lambda^a_{ij}(t)$ para diferenciarlo del parámetro utilizado en N-ALLIANCE con mismo nombre. Se debe ajustar $\delta_{ij}(t)$ y $\lambda_{ij}(t)$ según las siguientes expresiones:

$$\delta_{ij}(t) = \begin{cases} \min_k \delta\_\mathrm{slow}_{\mathrm{ij}}(k,t) & if\left(comm\_received\left(i,k,j,t-\tau_i,t\right)=1\right) \\ & and\left(comm\_received\left(i,k,j,0,t-\phi_{ij}(k,t)\right)=0\right) \\ \delta\_\mathrm{fast}_{\mathrm{ij}}(t) & \text{en otro caso} \end{cases}$$

$$\lambda_{ij}(t) = \begin{cases} \psi_{ij}(t) & if\ (\exists x.commReceived(i,x,j,t-\tau_i,t) = 1) \\ \lambda_{ij}^{a}(t) & \text{en otro caso} \end{cases}.$$

4.2.3. Implementación de L-ALLIANCE sobre N-ALLIANCE

Al igual que antes no se puede reproducir exactamente la asignación realizada por L-ALLIANCE pues ésta incluye mecanismos como ser la inicialización de la motivación (*impatience_reset) y* la motivación de aburrimiento (*learned_robot_influence*) por lo que se trata de reproducir los mecanismos de manera fiel a la arquitectura original.

A la implementación de L-ALLIANCE utilizando N-ALLIANCE se denomina L/N-ALLIANCE.

Al centrarse en la elección de la siguiente tarea a ejecutar se debe ajustar adecuadamente el parámetro δ. Si se ajusta según la expresión θ/ξ, donde ξ determina el tiempo de espera antes de la activación del comportamiento motivacional, se debe entonces asignar valores pequeños a ξ para las tareas que deban ejecutar primero y valores grandes a las tareas deban ejecutar últimas.

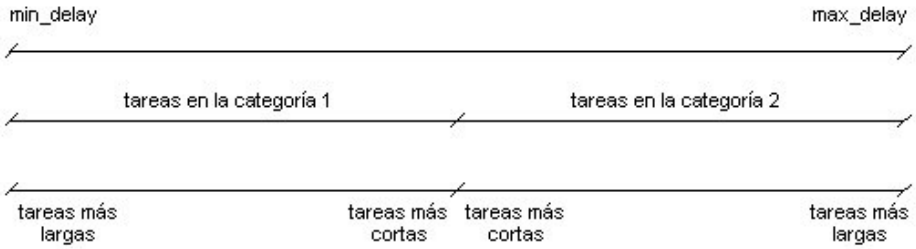


Figura 6. Valor de ξ por tareas para L-ALLIANCE

De manera inversa a la propuesta en L-ALLIANCE, se define un parámetro para cuando hay robots trabajando en la tarea y otro para cuando no hay ningún robot trabajando en la tarea, estos parámetros se denominan *ξ_slow* y *ξ_fast* respectivamente.

Se define el intervalo de tiempo [*min_delay,max_delay*] el cual determina el tiempo mínimo y máximo que puede pasar un robot ocioso sobre tareas que no están siendo realizadas por ningún otro robot, se puede hacer variar *ξ_fast* en dicho intervalo dependiendo de la estrategia que se desea implementar.

En la Figura 6, se muestra el intervalo [*min_delay,max_delay*] en el cual se hace variar *ξ_fast*, y se muestra como se divide este intervalo según la categoría y el tiempo de ejecución de la tarea, de manera de implementar la estrategia propuesta en L-ALLIANCE. Esta división del intervalo garantiza que todas las tareas pertenecientes a la categoría 1 tienen un menor valor de *ξ_fast* que los que toman las tareas de la categoría 2, y dentro de la categoría 1 se ordenan en orden decreciente de tiempo de ejecución, y dentro de la categoría 2 se ordenan en orden creciente de tiempo de ejecución. Esto permite implementar la estrategia preferida para elección de tareas.

Esta asignación se logra utilizando las siguientes ecuaciones:

$$high = \max_{kj} taskTime_i(k, j, t)$$

$$low = \min_{kj} taskTime_i(k, j, t)$$

$$mid\_delay = \frac{max\_delay - min\_delay}{2}$$

$$\xi_{1ij}(t) = \frac{taskTime_i(k, j, t) - low}{high - low}$$

$$\xi_{2ij}(t) = \frac{taskTime_i(k, j, t) - high}{low - high}$$

$$\xi\_fast_{ij}(t) = \begin{cases} \xi_{1ij}(t) * (mid\_delay - min\_delay) + min\_delay & if\ (taskCategory_{ij}(t) = 2) \\ \xi_{2ij}(t) * (max\_delay - mid\_delay) + mid\_delay & en\ otro\ caso \end{cases}$$

El valor de *ξ_slow* es ajustado de la misma forma que en L-ALLIANCE, esto es, tomando el valor de $\phi_{ij}(k,t)$.

Luego el valor de δ definido en la arquitectura N-ALLIANCE se define de manera análoga a la utilizada en ALLIANCE según la siguiente ecuación:

$$\delta_{ij}(t) = \begin{cases} \min_k \dfrac{\theta}{\xi\_\text{slow}_{ij}(k,t)} & if\ \left(comm\_received\left(i,k,j,t-\tau_i,t\right)=1\right) \\ & and\ \left(comm\_received\left(i,k,j,0,t-\phi_{ij}(k,t)\right)=0\right) \\ \dfrac{\theta}{\xi\_\text{fast}_{ij}(t)} & \text{en otro caso} \end{cases}.$$

Sólo resta ajustar el parámetro utilizado para el asentimiento en N-ALLIANCE, λ, el cual se ajusta según la siguiente ecuación:

$$\lambda_{ij}(t) = \begin{cases} \psi_{ij}(t) & if\ (\exists x.commReceived(i,x,j,t-\tau_i,t)=1) \\ \lambda^{a}_{ij}(t) & \text{en otro caso} \end{cases},$$

donde λ y ψ se ajustan exactamente de la misma manera en que fueron ajustados en L-ALLIANCE.

4.2.4. Finalización de N-ALLIANCE

Parker prueba el teorema de terminación de ALLIANCE [Parker1998] el cual asegura que dadas ciertas condiciones, un conjunto de robots controlados por la arquitectura ALLIANCE culminarán exitosamente la misión asignada si existe alguna forma de realizarla. A continuación se definen términos y condiciones necesarias que debe cumplir el equipo, para asegurar la finalización de la misión se refiere a la condición que se define a continuación.

Definición 1: Capacidades relevantes del robot r_i para la misión (goal-relevant capabilities) (GRC). Las capacidades relevantes del robot r_i para la misión están dadas por el siguiente conjunto:

$$GRC_i = \left\{ a_{ij} \middle| h_i(a_{ij}) \in T \right\},$$

donde *T* es el conjunto de tareas requeridas por la misión.

Condición 1: Task Coverage suficiente,

$$\forall(task \in T)(taskCoverage(task) \geq 1).$$

Condición 2: Progreso cuando trabaja (Progress When Working) (PWW).

PWW asegura que: cuando un robot r_i activa el comportamiento motivacional asociado a la tarea *w* pueden ocurrir dos cosas (1) el robot r_i mantiene el comportamiento motivacional activo por un período de tiempo finito, ε, de forma que el tiempo requerido para la tarea *w* se reduce en al menos δ unidades de tiempo, $\delta \geq 1$, o (2) el robot experimenta una falla respecto de la tarea *w*.

Definición 2: Para un conjunto de robots activo, *R,* se debe cumplir:

$$\forall(r_i \in R).\forall(a_{ij} \in GRC_i).\forall t\left[\left(\delta_{ij}(t) \geq 1\right) \wedge \left(\lambda_{ij}(t) > \varepsilon\right) \wedge \left(\theta\, finito\right)\right].$$

N-ALLIANCE elimina y simplifica varios de los mecanismos utilizados en ALLIANCE, por lo que la demostración de finalización para N-ALLIANCE se basa en la demostración para ALLIANCE.

Teorema 1: Sea *R* un conjunto de robots activos, y *M* la misión a ser resulta por *R*. Si se cumple la condición 1 y la condición 2. Pueden ocurrir dos cosas (1) ALLIANCE hace que *R* cumpla con *M*, o (2) ocurre una falla en un robot. Más aún, si ocurre una falla en un robot r_f las únicas tareas que no son ejecutadas son un subconjunto de las tareas que sabe realizar r_f unión las tareas que dependen de las capacidades de r_f.

Demostración

De la ecuación de actualización de la motivación en N-ALLIANCE, se deduce que en tiempo *t* la motivación del robot r_i, $m_{ij}(t)$, para ejecutar el comportamiento de alto nivel a_{ij}, (1) vale cero, o (2) se incrementa según $\delta_{ij}(t)$. El caso (1) puede ocurrir debido al menos a una de cuatro situaciones, (a) la información sensorial determina que el comportamiento de alto nivel no debe ejecutarse, (b) otro comportamiento de alto nivel se activo, (c) el robot cede la tarea o (d) el robot finalizó la tarea. En el caso (a) puede ocurrir que el robot haya completado la tarea o que se deba a una falla en el robot. El caso (b) si el robot r_i activa otro comportamiento de alto nivel a_{ik}, en algún momento va a completar la ejecución de a_{ik}, permitiéndole activar el comportamiento de alto nivel a_{ij}. Los casos (c) y (d) no ocurren pues el comportamiento de alto nivel aún no fue activado. Por otro lado, como r_i pertenece a un conjunto de robots activos se cumple que $\delta_{ij}(t) \geq 1$, entonces un robot ocioso tiene una motivación estrictamente creciente. Como θ es finito en algún momento la motivación del comportamiento de alto nivel lo superará, haciendo que el robot r_i active el comportamiento de alto nivel a_{ij}.

Por lo que, que todo comportamiento de alto nivel a_{ij} será ejecutado por el robot r_i siempre que la información sensorial asociada a éste indique debe ejecutarse.

Parte 1 (N-ALLIANCE tiene éxito o falla un robot)

Cuando un robot ejecuta una tarea, *w*, puede ocurrir al menos una de las siguientes cuatro situaciones:

1. El robot r_i cede la tarea *w.*
2. El robot r_i continúa ejecutando la tarea *w*.
3. El robot r_i finaliza la tarea *w*.
4. El robot r_i completa la tarea *w*.

En el caso 1, como es un conjunto activo de robots y se cumple la condición 2, el tiempo de ejecución de la tarea se reduce en al menos δ unidades de tiempo. Como el tiempo de ejecución de una tarea es finito, ésta será completada en tiempo finito. En el caso 2, el robot continuará ejecutando la tarea hasta que se cumpla el caso 1, 3 o 4. En el caso 3, el robot volverá a activar la tarea w en el futuro, hasta que se cumpla el caso 4, lo cual ocurre en tiempo finito pues el tiempo necesario para completar una tarea es finito. En el caso 4, el sistema sensorial indica que la tarea ya no debe ejecutarse por lo que el robot pasará a ejecutar otra tarea.

Por esto, para cada tarea no completada, (1) un robot capacitado para ejecutarla la ejecuta tantas veces como sea necesario para completarla, o (2) todos los robots designados para ejecutarla fallan.

Parte 2 (Las tareas incompletas dependen de las capacidades de los robots que fallan)

Sea F el conjunto de robots que fallan durante la misión y A_F el conjunto formado por las tareas que los robots en F saben realizar unión las tareas que dependen de éstas últimas.

Primero se muestra que si una tarea no está en A_F será exitosamente completada. Sea w una tarea que no pertenece a A_F, como se cumple la condición 1 y el conjunto de robots es activo, debe haber un robot en el equipo que puede ejecutar exitosamente la tarea w. Luego, mientras la tarea permanezca incompleta este robot activa el comportamiento de alto nivel asociado a la tarea, como se cumple la condición 2 esa tarea será completada en tiempo finito. Por esto, todas las tareas no dependientes de las capacidades de los robots que fallan serán completadas en N-ALLIANCE.

A continuación se muestra que cuando una tarea no es completada debe pertenecer a A_F. Sea w la tarea que no fue completada al finalizar la misión. Asumamos que w no pertenece a A_F. Se demostró en la parte 1 que todas las tareas que no están en A_F son completadas. Por lo que, la tarea w debe pertenecer a A_F. •

Este resultado es muy importante pues como se indico antes Parker al definir L-ALLIANCE modifica el formalismo de ALLIANCE sin probar la finalización para L-ALLIANCE. Se puede asegurar que las implementaciones que se proponen en este capítulo y todas las que se construyan sobre N-ALLIANCE finalizaran.

5. TC/N-ALLIANCE y CR/N-ALLIANCE

5.1. Introducción

En este capítulo se presentan dos estrategias para mejorar la asignación realizada por L-ALLIANCE. Se ilustran las ideas a partir de las cuales nacen estas estrategias y las ventajas obtenidas al usarlas. Luego se definen formalmente las expresiones utilizadas para actualizar los parámetros de N-ALLIANCE de forma de implementar dichas estrategias.

En la sección 5.2 se presentan las razones principales que motivan el desarrollo de estas nuevas estrategias y la sección 5.3 se definen y formalizan las estrategias permitiendo implementarlas sobre N-ALLIANCE.

5.2. Motivación

Las dos estrategias que se definen a continuación surgen para resolver deficiencias ó simplificaciones realizadas en la asignación automática utilizando L-ALLIANCE. Estas estrategias se denominan TC/N-ALLIANCE (Task Coverage N-ALLIANCE) y CR/N-ALLIANCE (Cantidad de Robots N-ALLIANCE).

En la primer estrategia se toman las observaciones realizadas en el Capítulo 3 que proponen tener en cuenta el task coverage para realizar una mejor asignación de tareas, teniendo en cuenta que las tareas que sólo unos pocos robots saben realizar deben ser elegidas primero por dichos robots mientras el resto del equipo se encarga de realizar otras tareas simultáneamente.

En la segunda estrategia se propone que los robots se distribuyan las tareas uniformemente. L-ALLIANCE y sus estrategias de impaciencia y asentimiento no permiten, al menos al principio de la misión, que los robots trabajen juntos en una misma tarea pues la impaciencia es ajustada de forma de no molestar a los robots que están realizando una determinada tarea. La estrategia CR/N-ALLIANCE permite que todos los robots sean rápidamente asignados a tareas teniendo en cuenta sus aptitudes para realizarla y la cantidad de robots que están trabajando en la misma.

5.3. Nuevas Estrategias

5.3.1. TC/N-ALLIANCE

Parker define en L-ALLIANCE el task coverage de una tarea como la cantidad de robots que saben realizarla, formalmente es definida de la siguiente manera:

$$taskCoverage(task) = \sum_{k \in R} \sum_{j \in A_k} \begin{Bmatrix} 1 & h_k(a_{kj}) = task \\ 0 & en\ otro\ caso \end{Bmatrix}.$$

En TC/N-ALLIANCE, se proponen extensiones a la estrategia propuesta en L-ALLIANCE, incorporando el valor de task coverage para actualizar automáticamente los parámetros.

Una de las carencias clara en que cae la estrategia planteada en L-ALLIANCE es no considerar el task coverage en la elección de las tareas. Parece razonable pensar que las tareas con menor task coverage deben elegirse primero evitando de esta forma que dichas tareas sean ejecutadas al final por unos pocos robots quedando el resto del equipo sin posibilidades de actuar. En otras palabras, se desea que las tareas con task coverage bajo sean elegidas primero y de esta forma ejecutarlas en paralelo con la actividad de otros robots. En el Capítulo 3 se mostró un ejemplo que utiliza el

mecanismo propuesto por L-ALLIANCE y al no tener en cuenta el task coverage no logra una asignación óptima.

La estrategia que se define no utiliza la definición de task coverage dada por Parker pues la misma no permite adaptarse a cambios en la configuración del equipo. Dicha definición es sustituida por una definición más dinámica que logra una mejor adaptación. La definición formal de la función utilizada es:

$$taskCoverage_{ij}(t) = \sum_{k \in robotsPresents(i,t)} \sum_{a_{ik} \in A_{ik}(t)} \left\{ \begin{matrix} 1 & h(a_{ik}) = h(a_{ij}) \\ 0 & en\ otro\ caso \end{matrix} \right\},$$

donde A_{ik} es el conjunto de comportamientos que el robot *i* conoce del robot *k*, $A_{ik} \subset A_i$. Esta definición permite en todo momento conocer el task coverage real, que puede cambiar por la inserción o eliminación de un miembro del equipo, o el agregado de un nuevo comportamiento de alto nivel.

Al centrarse en la elección de la siguiente tarea a ejecutar se debe ajustar adecuadamente el parámetro δ. Si se ajusta según la expresión θ/ξ, ξ determina el tiempo de espera antes de la activación del comportamiento motivacional. Se debe entonces asignar valores pequeños a ξ para las tareas que deban ejecutar primero y valores grandes a las tareas que deban ejecutar últimas.

Se define el intervalo de tiempo [*min_delay,max_delay*] el cual determina el tiempo mínimo y máximo que puede pasar un robot ocioso. Así se puede hacer variar ξ en dicho intervalo dependiendo de la estrategia que se desea implementar.

En esta estrategia, al igual que antes, cada robot divide las tareas en dos categorías

(1) Las tareas que sabe realizar mejor que cualquier otro robot y no está siendo realizada por otro robot.

(2) El resto de las tareas.

Luego se elige primero dentro de la categoría (1) y cuando no quedan más tareas en esta categoría elige dentro de la categoría (2). Si hubiera más de una tarea en la categoría (1) elige primero según la tarea con menor *taskCoverage* y si hubiera varias con igual *taskCoverage* elige la tarea más larga. Luego si hubiera más de una tarea en la categoría (2) elige primero según la tarea con menor *taskCoverage* y si hubiera varias con igual *taskCoverage* elige la tarea más corta.

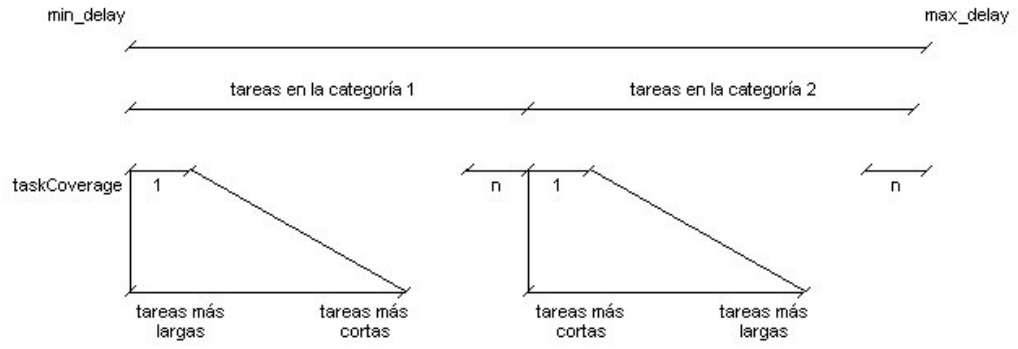


Figura 7. Valor de ξ por tareas cuando se trabaja sin interferencia

En la Figura 7 se aprecia el valor que debe tomar ξ para implementar la estrategia propuesta, en función del tiempo de ejecución de la tarea, el task coverage y la categoría a la que pertenece.

Para lograr dicha asignación primero se define la función ξ_{1ij} que asigna valores altos, cercanos a uno, a las tareas que el robot demora más en ejecutar y valores chicos, cercanos a cero, a las tareas que demora menos en ejecutar. Esta función se define de la siguiente manera:

$$high = \max_{kj} taskTime_i(k, j, t)$$
$$low = \min_{kj} taskTime_i(k, j, t) \quad .$$
$$\xi_{1ij}(t) = \frac{taskTime_i(i, j, t) - low}{high - low}$$

De manera análoga, se define la función ξ_{2ij} que asigna valores altos, cercanos a uno, a las tareas que el robot demora menos en ejecutar y valores chicos, cercanos a cero, a las tareas que demora más en ejecutar. Esta función se define de la siguiente manera:

$$\xi_{2ij}(t) = \frac{taskTime_i(i, j, t) - high}{low - high}.$$

Luego se define un par de funciones que asignan valores altos, cercanos a uno, a las tareas con mayor *taskCoverage* y valores chicos, cercanos a cero, a las tareas con menor *taskCoverage*. Estas funciones se diferencian en el orden establecido para iguales *taskCoverage*, la función ξ'_{1ij} define un orden creciente según tiempos de ejecución y la función ξ'_{2ij} un orden decreciente según tiempos de ejecución. Este par de funciones se definen de la siguiente manera:

$$\xi'_{hij}(t) = \frac{\xi_{hij}(t) + taskCoverage_{ij}(t)}{\#robotsPresents_i(t) + 1}, h \in \{1, 2\}.$$

Ahora sólo resta ajustar los valores de ξ_{ij} de forma que se respete el orden global de la estrategia y el tiempo mínimo y máximo de espera. Esto se logra definiendo la función ξ_{ij} de la siguiente manera:

$$\xi_{ij}(t) = \begin{cases} \xi'_{1ij}(t) * (mid\_delay - min\_delay) + min\_delay & if\ (taskCategory_{ij}(t) = 1) \\ \xi'_{2ij}(t) * (max\_delay - mid\_delay) + mid\_delay & en\ otro\ caso \end{cases} .$$
$$where\ mid\_delay = min\_delay + (max\_delay - min\_delay)/2$$

El resto de los parámetros se ajustan de igual forma que en L-ALLIANCE.

En el Capítulo 6 se presentan los resultados obtenidos con esta estrategia y se compara con la asignación producida por L-ALLIANCE.

5.3.2. CR/N-ALLIANCE

Hasta el momento no se hay definido ningún mecanismo de actualización de parámetros que tomen en cuenta la cantidad de robots que trabajan en una tarea, ni tampoco las tareas que deben ejecutarse más de una vez durante la misma misión (tareas repetitivas). La estrategia que se presenta a continuación responde a algunas de las preguntas planteadas en los Capítulos 3 y 4.

Esta estrategia es denominada CR/N-ALLIANCE pues incorpora la cantidad de robots que se encuentran trabajando en la tarea para actualizar automáticamente los parámetros.

CR/N-ALLIANCE apunta a que los robots se dividan las tareas de forma uniforme de acuerdo al rendimiento esperado de ejecución, principalmente pensando en misiones donde la cantidad de robots sea superior a la cantidad de tareas, pudiendo en este caso trabajar varios robots simultáneamente sobre una misma tarea en lugar de quedarse ociosos.

ALLIANCE en su definición de impaciencia determina que en caso de existir algún robot trabajando en una tarea, el comportamiento motivacional asociado a dicha tarea en el resto de los robots debe impacientarse lentamente pero no permite impacientarse inversamente proporcional a la cantidad de robots que trabajan.

En esta estrategia cada robot divide las tareas en dos categorías

(1) Las tareas que sabe realizar mejor que cualquier otro robot.

(2) El resto de las tareas.

Luego se elige primero dentro de la categoría (1) y cuando no quedan más tareas en esta categoría elige dentro de la categoría (2). Si hubiera más de una tarea por categoría elige según la tarea para la cual hay menor cantidad de robots trabajando y está más apto para ejecutarla.

Una diferencia clara con el resto de las estrategias es que la categoría 1 no toma en cuenta si hay robots trabajando.

Antes de definir el mecanismo de actualización automático para los parámetros es necesario realizar algunas definiciones previas. Se define primero la función $working_{ij}(k,t)$ que indica si el robot r_k está trabajado en la tareas $h(a_{ij})$ según el robot r_i. Formalmente la definición para esta función es la siguiente:

$$\text{working}_{ij}(k,t)=\begin{cases}1 & \textit{si para el robot i el robot k está realizando la tarea } h(a_{ij}) \\ 0 & \textit{en otro caso}\end{cases}.$$

Luego se define una función que indique la cantidad de robots que trabajan en una determinada tareas, la misma es definida formalmente de la siguiente manera:

$$\text{robotsWorking}_{ij}(t)=\sum_{k\in robotsPresents(i,t)} \text{working}_{ij}(k,t)\ .$$

Estas dos funciones son computadas por los monitores de actividad sin necesidad de grandes modificaciones.

Luego de esta introducción y algunas definiciones previas, se define el mecanismo de actualización automático siguiendo el criterio de asignación para la impaciencia que se muestra en la Figura 8. Este mecanismo permite seleccionar primero las tareas que no están siendo realizadas pon ningún robot, si hay varias tareas en estas condiciones elige según la propuesta anterior, esto es, eligiendo la tarea con menor task coverage, en la categoría 1 con mayor tiempo de ejecución y en caso de no existir elige en la categoría 2 con menor tiempo de ejecución. Si no existe ninguna tarea libre selecciona entre las tareas con un robot trabajando, y así sucesivamente hasta llegar a las tareas donde todos los robots están trabajando.

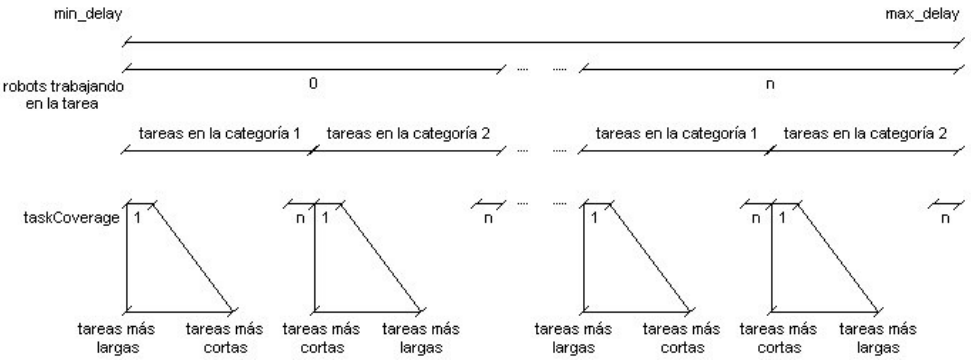


Figura 8. Valor de ξ por tareas cuando se trabaja con interferencia

Esta estrategia se propone con el objetivo de distribuir uniformemente las tareas entre los robots.

Al igual que antes, para determinar la elección de la siguiente tarea a ejecutar se debe ajustar adecuadamente el parámetro δ. Si se ajusta según la expresión θ/ξ, ξ determina el tiempo de espera antes de la activación del comportamiento motivacional. Se debe entonces asignar valores pequeños a ξ para las tareas que deban ejecutar primero y valores grandes a las tareas que deban ejecutar últimas.

El mecanismo de actualización propuesto aquí debe asignar valores mínimos a ξ para las tareas que no están siendo ejecutadas por ningún robot, pertenezcan a la categoría 1 con task coverage uno y tiempo de ejecución alto, y valores máximos a ξ para las tareas más que están siendo ejecutadas por *n* robots, pertenezcan a la categoría 2 con task coverage igual a *n* y tiempo de ejecución alto.

Se define el intervalo de tiempo [*min_delay,max_delay*] el cual determina el tiempo mínimo y máximo que puede pasar un robot ocioso. De este modo, se puede hacer variar ξ en dicho intervalo de forma de implementar el mecanismo propuesto.

Para esto se define una función que asigna valores altos para ξ_{1ij}, cercanos a uno, a las tareas que el robot demora más en ejecutar y valores chicos, cercanos a cero, a las tareas que demora menos en ejecutar. Esta función se define de la siguiente manera:

$$high = \max_{kj} taskTime_i(k, j, t)$$

$$low = \min_{kj} taskTime_i(k, j, t) \quad .$$

$$\xi_{1ij}(t) = \frac{taskTime_i(k, j, t) - low}{high - low}$$

De manera análoga, se define una función que asigna valores altos para ξ_{2ij}, cercanos a uno, a las tareas que el robot demora menos en ejecutar y valores chicos, cercanos a cero, a las tareas que demora más en ejecutar. Esta función se define de la siguiente manera:

$$\xi_{2ij}(t) = \frac{taskTime_i(k, j, t) - high}{low - high}.$$

Luego se define un par de funciones que asignan valores altos, cercanos a uno, a las tareas con mayor *taskCoverage* y valores chicos, cercanos a cero, a las tareas con menor *taskCoverage*.

Estas funciones se diferencian en el orden establecido para iguales *taskCoverage*, la función ξ_{1ij} define un orden creciente según tiempos de ejecución y la función ξ_{2ij} un orden decreciente según tiempos de ejecución. Este par de funciones se definen de la siguiente manera:

$$\xi'_{hij}(t) = \frac{\xi_{hij}(t) + taskCoverage_{ij}(t)}{\#robotsPresents_i(t)+1}, h \in \{1,2\}.$$

Se define una función que asegura la elección de tareas de forma que se opte primero por las tareas más largas de la categoría 1 y luego dentro de la categoría 2 eligiendo primero las tareas más cortas. Con este fin se define la siguiente función:

$$\xi''_{ij}(t) = \begin{cases} \dfrac{\xi'_{1ij}(t)}{2} & if\ (taskCategory_{ij}(t) = 1) \\ \dfrac{\xi'_{2ij}(t)+1}{2} & en\ otro\ caso \end{cases}.$$

Ahora sólo resta ajustar los valores de ξ_{ij} de forma que se respete el orden global de la estrategia y el tiempo mínimo y máximo de espera. Esto se logra definiendo la función ξ_{ij} de la siguiente manera:

$$\xi_{ij}(t) = \frac{\xi''_{ij} + robotsWorking_{ij}(t)}{robotsPresents(i,t)+1} * (max\_delay - min\_delay) + min\_delay.$$

Es necesario definir ahora como se ajusta el valor de λ teniendo en cuenta que esta nueva estrategia permite una colaboración más intensa entre los robots, esto lleva a tener varios robots trabajando en la misma tarea lo cual, en general, determina que los tiempos de ejecución individuales se incrementen. Es razonable ajustar λ de manera proporcional a la cantidad de robots que ejecutan la tarea. Para esto se definen dos parámetros, *lambda_min* y *lambda_max*, que determinan los valores mínimos y máximos que puede tomar λ, luego se ajusta su valor de acuerdo a la siguiente expresión:

$$\lambda_{ij}(t) = \frac{robotsWorking_{ij}(t)}{robotsPresents(i,t)} * (lambda\_max_{ij}(t) - lambda\_min_{ij}(t)) + lambda\_min_{ij}(t).$$

En la práctica se obtuvieron buenos resultados ajustando *lambda_min* igual al tiempo requerido para realizar la tarea y *lambda_max* igual a dos veces dicho tiempo, lo cual determina la siguiente expresión de actualización automática:

$$\lambda_{ij}(t) = taskTime_i(i,j,t) * \left(\frac{robotsWorking_{ij}(t)}{robotsPresents(i,t)} + 1 \right).$$

En el Capítulo 6 se presentan los resultados obtenidos con esta estrategia y se compara con la asignación producida por L/N-ALLIANCE y por TC/N-ALLIANCE.

6. Experimentos y Resultados

6.1. Introducción

En este capítulo se presenta el caso de estudio elegido para realizar los experimentos con el simulador YAKS, la definición del mundo utilizado y los resultados obtenidos luego de ejecutar las distintas estrategias. En el Apéndice A se describe el simulador de robots Khepera YAKS y las modificaciones realizadas para implementar el caso de estudio.

En este punto se trata de comparar el mecanismo automático propuesto por Parker contra las estrategias propuestas en este trabajo. Para esto, se analizan los resultados obtenidos en ejecuciones que utilizan el simulador YAKS, y por otro lado las ejecuciones obtenidas variando los parámetros de la misión sin interactuar con el simulador YAKS, generando misiones virtuales. Tanto las misiones sobre el simulador como las misiones virtuales pretenden comparar el rendimiento de L-ALLIANCE con las nuevas estrategias propuestas. En el primer caso se pretende mostrar el efecto de la interferencia entre los robots. Mientras que los experimentos que no utilizan el simulador pretenden obtener mayor cantidad de corridas mediante la ejecución de misiones virtuales. La ejecución de misiones virtuales permite recabar mucha información, variar rápidamente los parámetros de la misión y permitir repetir la misma misión.

En este capítulo también se define una métrica que permite evaluar de manera más justa el rendimiento de las misiones ejecutadas en el simulador.

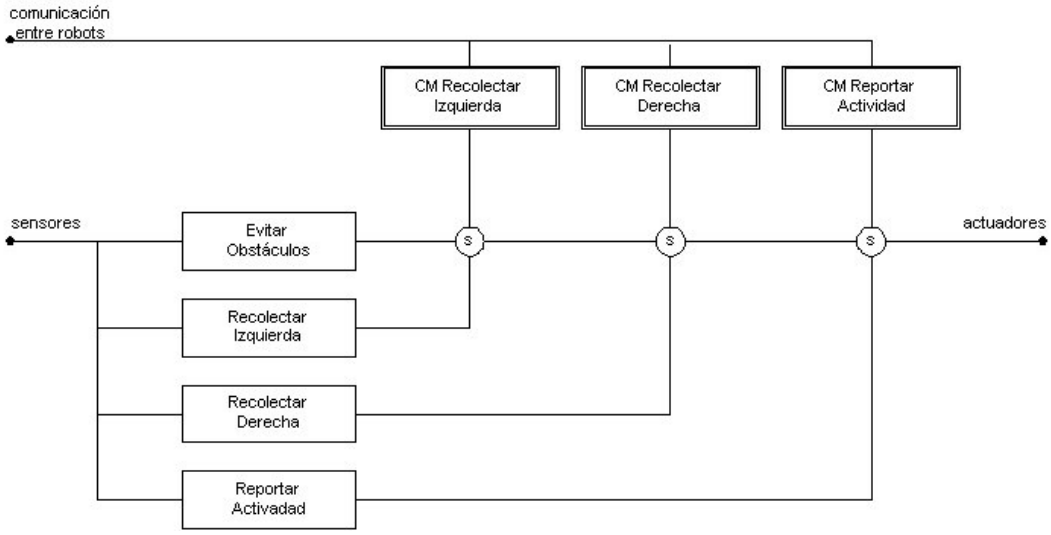


Figura 9. Instancia de ALLIANCE para empujar objetos.

6.2. Caso de estudio

Se utilizo para probar la efectividad y el rendimiento de las nuevas estrategias el problema de recolección de objetos. Este problema es utilizado en [Parker1998] para verificar las bondades de ALLIANCE y es un caso de estudio adecuado para probar las nuevas estrategias definidas. El problema planteado propone que los robots que componen al equipo resuelvan una misión compuesta por tres tareas, recolectar objetos en una zona ubicada a la izquierda del mundo (Comportamiento Recolectar Izquierda) (CRI), recolectar objetos en una zona ubicada a la derecha del mundo

(Comportamiento Recolectar Derecha) (CRD) y reportar cada cierto tiempo la actividad del equipo (Comportamiento Reporte de Actividad) (CRA). Se define la misión para el caso de estudio como el conjunto de tareas *T={CRI, CRD, CRA}*.

En el Capítulo 7 se detalla la implementación que se realizó sobre el simulador YAKS de los comportamientos evitar obstáculos, recolectar objetos y reportar actividad. En la Figura 9 se muestra la arquitectura ALLIANCE utilizada para resolver el problema elegido. Se puede apreciar en la misma los tres comportamientos de alto nivel que componen la misión y asociados a ellos los comportamientos motivacionales que controlan su activación, aparece un comportamiento de bajo nivel o reflejo asociado a evitar obstáculos que se activa cuando todos los comportamientos de alto nivel están hibernando.

Para ejecutar el caso de estudio propuesto es necesario instanciar los comportamientos, los comportamientos motivacionales, los supresores de actividad y el servicio de comunicación. Todos estos elementos luego son utilizados por la arquitectura N-ALLIANCE para lograr el comportamiento de equipo. Se pueden consultar el diseño de la arquitectura, los componentes indicados y el servicio de comunicación en el Apéndice B.

6.3. Experimentos y Resultados

Para analizar las estrategias propuestas se creo un mundo similar al propuesto en [Parker1998] en el simulador YAKS. En la Figura 10 se puede apreciar el mundo utilizado para realizar todas las experiencias que se detallan más adelante en esta sección. En el mismo se muestran las zonas de interés como círculos verdes.

En la parte superior izquierda del mundo se puede ver la zona izquierda asignada para recolectar objetos, en la misma aparecen diez objetos a ser recolectados por los robots cuando activen el comportamiento RecolectarIzquierda y dos de los robots que componen al equipo.

En la parte superior derecha del mundo se puede observar la otra zona de recolección con diez objetos a ser recolectados por los robots cuando activen el comportamiento RecolectarDerecha. En el centro del mundo se encuentra la zona de descarga, o home, hacia donde los robots deben trasladar los objetos desde las zonas de recolección a la zona de descarga. Por último, en el centro inferior del mundo se puede apreciar la zona de reporte, hacia donde los robots se deben trasladar hasta dicha zona para emitir el reporte de actividad. En este caso se muestra el mundo cuando el equipo está formado por tres robots, dos de ellos ubicados en la zona de recolección izquierda, y el tercero se puede apreciar sobre el vértice inferior izquierdo del mundo. Los robots son ubicados al azar dentro del mundo y los objetos son ubicados uniformemente pero dentro de sus respectivas zonas de recolección.

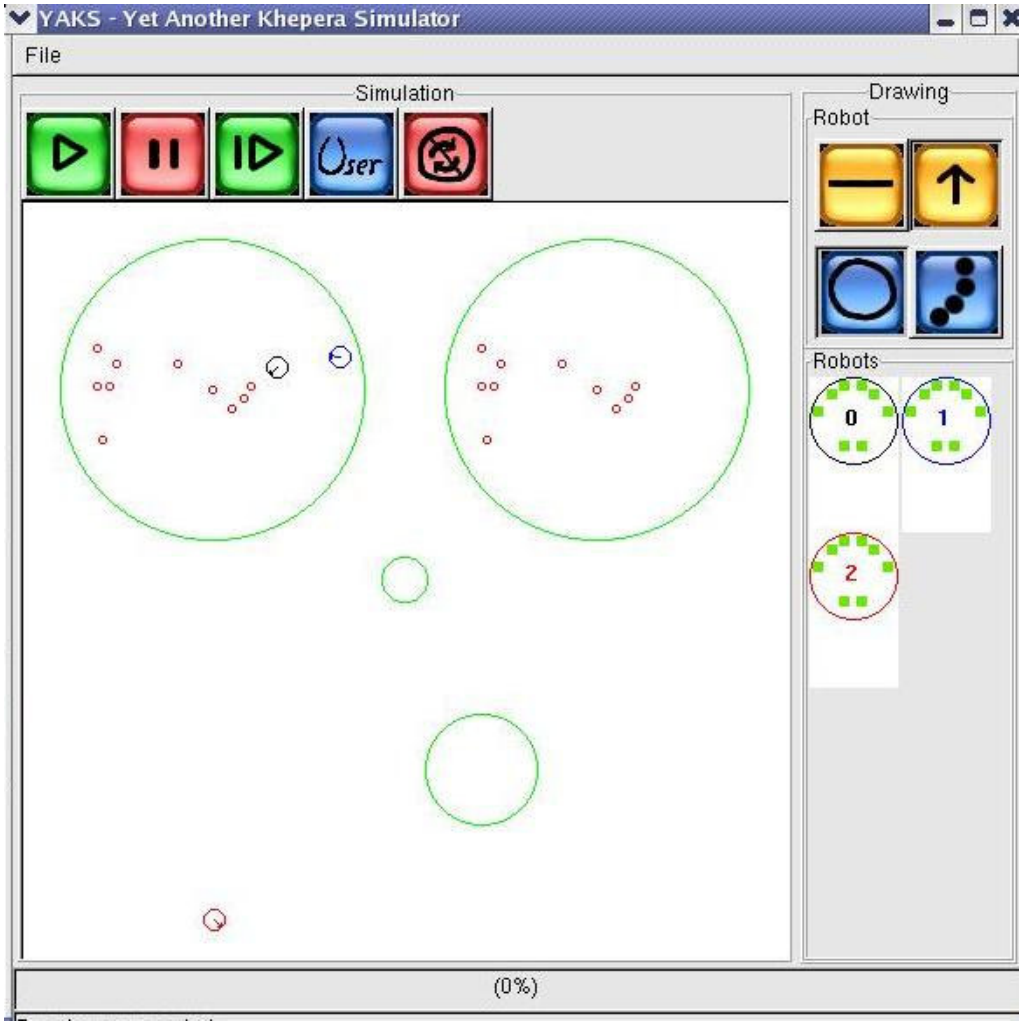


Figura 10. Configuración inicial del mundo para el caso de estudio.

En la Figura 11 se puede apreciar el mundo cuando el equipo de robots finalizó la misión recolectando los objetos en ambas zonas. Como no quedan más objetos para recolectar los robots no tienen otra cosa que hacer que reportar la actividad por lo que todos terminan en la zona de reporte emitiendo un informe de la actividad.

El uso de simuladores es muy criticado en los artículos de robótica móvil, principalmente por utilizar modelos reducidos del mundo, que llevan al fracaso de las aplicaciones en robótica realizadas sobre simuladores cuando son probadas en un mundo real. En este, trabajo sólo se trabajó sobre simuladores principalmente por las dificultades al momento de disponer de robots reales. Cuando se trabaja con simuladores se obtienen varias ventajas entre las que se destacan las siguientes:

- es posible repetir exactamente el mismo experimento cuantas veces sea necesario.
- la fuente de energía de los robots es ilimitada, lo que permite trabajar continuamente sin necesidad de cargar sus baterías.
- no es necesario disponer de infraestructura para la investigación.
- no es necesario disponer de robots para investigar en esa área.

Pero no todo son ventajas, el atractivo de ver a un robot real trabajando o realizando una determinada tarea en un entorno real, es una sensación muy agradable y que llena de satisfacción al diseñador, pues en definitiva es ahí donde todos queremos que estén interactuando en un mundo real.

Para presentar los resultados se armaron las gráficas de activación que se muestran en las siguientes figuras. En las mismas se aprecia sobre el eje x el tiempo en segundos y sobre el eje y se indica qué tarea activa cada robot. En las primeras gráficas se muestran los tiempos de activación para tres robots utilizando diversas estrategias de selección de tareas y en la última gráfica se muestra un ejemplo con seis robots realizando las tareas propuestas. En las gráficas se muestra la activación de la tarea recolectar por la izquierda, recolectar por la derecha y reportar actividad, y en caso de que todos los comportamientos estén hibernando el comportamiento que se marca como activo es el ocio.

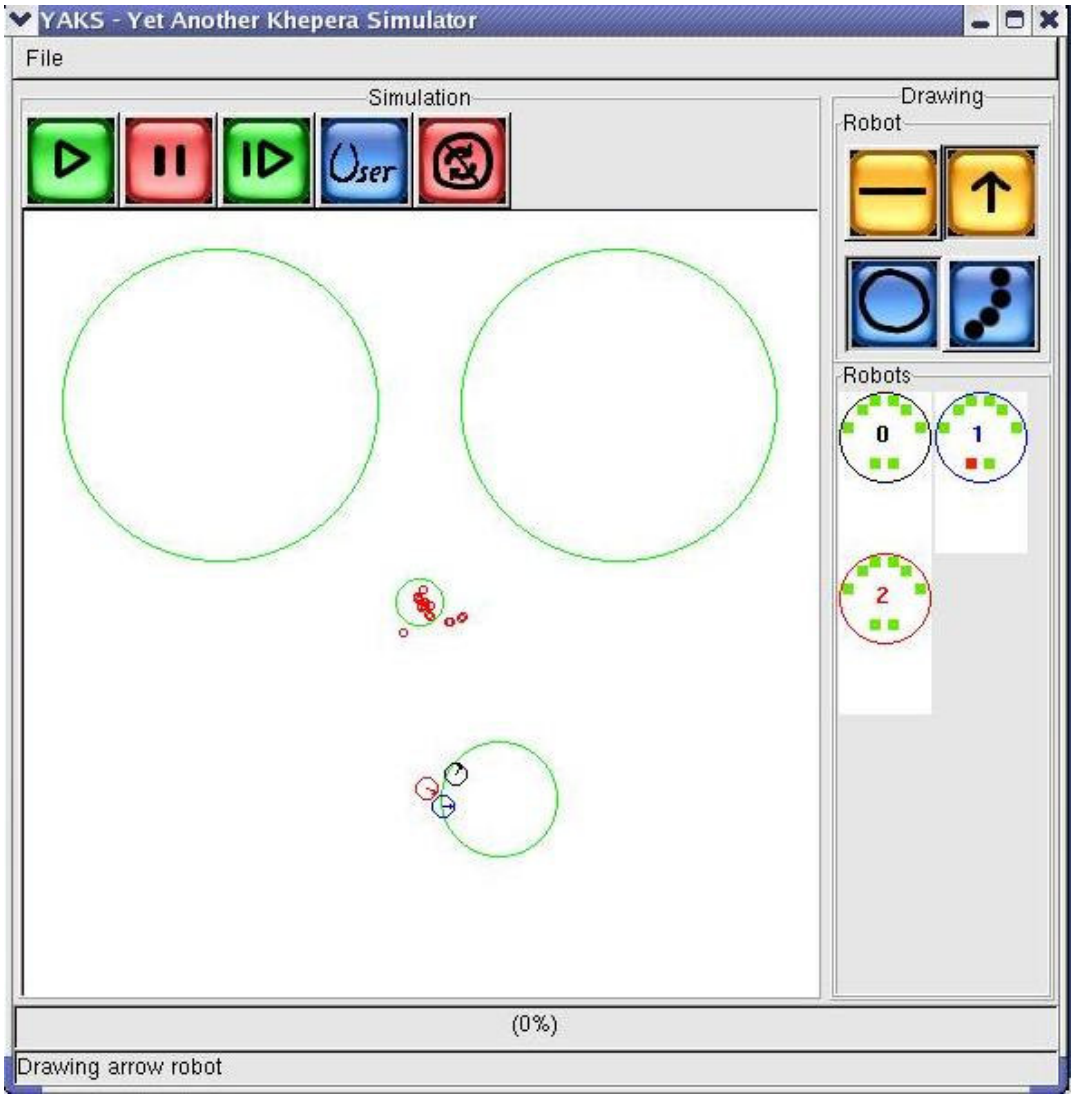


Figura 11. Estado final del mundo para el caso de estudio.

6.3.1. TC/N-ALLIANCE

Para esta estrategia se presenta primero una misión con tres robots homogéneos y luego una misión con tres robots heterogéneos.

6.3.1.1 Misión con tres robots homogéneos

En este caso se utilizan tres robots donde cada uno puede realizar cualquier tarea requerida por la misión, la definición para este caso es la siguiente:

- El conjunto de robot cooperativo, $R=\{r_{rojo}, r_{verde}, r_{azul}\}$
- Los comportamientos de alto nivel, $A_{rojo}=\{a_{rojo\ 1}, a_{rojo\ 2}, a_{rojo\ 3}\}$, $A_{verde}=\{a_{verde\ 1}, a_{verde\ 2}, a_{verde\ 3}\}$, $A_{azul}=\{a_{azul\ 1}, a_{azul\ 2}, a_{azul\ 3}\}$
- Las funciones de traducción entre comportamientos de alto nivel y tareas, $h_i(a_{i1})=CRI$, $h_i(a_{i2})=CRD$ y $h_i(a_{i3})=CRA$, $i \in \{rojo,verde,azul\}$

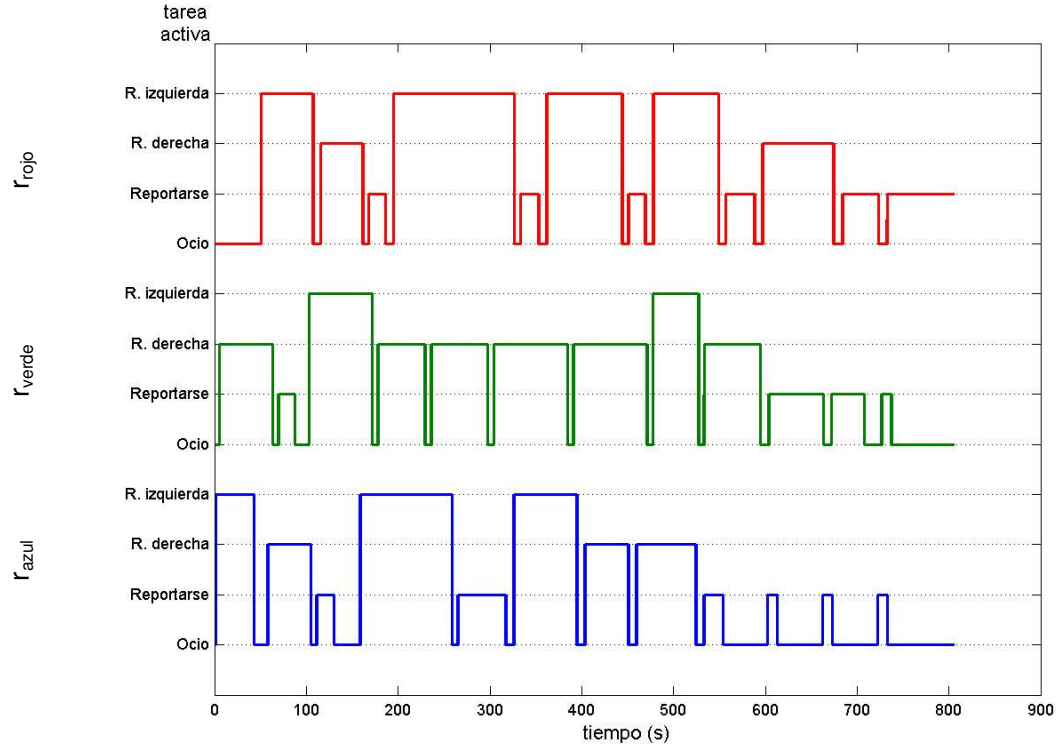


Figura 12. Asignación de tareas utilizando TC/N-ALLIANCE para un equipo homogéneo.

En la Figura 12 se muestra una asignación de tareas a robots realizada cuando todo el equipo se rige por las reglas de control de la estrategia TC/N-ALLIANCE.

El robot rojo elige durante la misión las siguientes tareas CRI, CRD, CRA, CRI, CRA, CRI, CRA, CRI, CRA, CRD, CRA y CRA. El robot verde elige durante la misión las siguientes tareas CRD, CRA, CRI, CRD, CRD, CRD, CRD, CRI, CRD, CRA, CRA y CRA. El robot azul elige durante la misión las siguientes tareas CRI, CRD, CRA, CRI, CRA, CRI, CRD, CRD, CRA, CRA, CRA y CRA.

La simulación del caso de estudio determina que cuando un robot explora una determinada zona de recolección y la exploración lo aleja del centro de alimento, este

debe asumir que no hay más alimento para recolectar. Esto se logra conectando a la información de percepción del comportamiento recolectar un sensor virtual que sea cero cuando se determina que no hay más objetos. El robot azul determina en tiempo 520 que no hay más objetos para recolectar por lo que el único comportamiento que puede activar es el de reporte de actividad. El robot verde determina en tiempo 600 que no hay más alimento para recolectar y el robot rojo en tiempo 680.

En general, la asignación fue uniforme, esto se puede ver en tiempo 80, 180, 270, y 350 donde cada uno de los robots está realizando una de las tareas. Esto ocurre principalmente por ser la cantidad de robots igual a la cantidad de tareas, existiendo también un componente de azar pues TC/N-ALLIANCE no trata de distribuir uniformemente los robots entre las tareas, y como se explicó antes esta estrategia no se comporta adecuadamente cuando las tareas son repetitivas, como lo son todas las tareas a realizar en este caso.

Cuando los robots comienzan a ejecutar, el azul elige la tarea recolectar izquierda, el verde la tarea recolectar derecha y el rojo queda ocioso permitiendo que sus compañeros realicen la tarea que eligieron.

El tiempo requerido por la misión es 680 segundos, lo cual por si solo no aporta toda la información necesaria para medir el rendimiento del equipo, esto se debe a que puede ocurrir que un robot al activar uno de los comportamientos de recolección cargue con más de un objeto. En la Tabla 6 se incluye un resumen de los datos más representativos de la misión que permiten sacar conclusiones de rendimiento respecto de otras estrategias. En la última fila de esta tabla se muestra el valor obtenido al dividir la duración de la misión entre la cantidad de tareas de recolección, esta medida es utilizada en lugar de la duración de la misión pues como se comentó antes a veces los robots cargan con más de un objeto.

Tabla 6. Resumen de datos para TC/N-ALLIANCE para un equipo homogéneo

Dato	Valor
Cantidad de robots	3
Cantidad de tareas	3
Equipo homogéneo	Si
Cantidad de ejecuciones de CRI (#CRI)	9
Cantidad de ejecuciones de CRD (#CRD)	11
Cantidad de ejecuciones de CRA	7
Duración de la misión	680
Duración de la misión/(#CRI+#CRD)	34

6.3.1.2 Misión con tres robots heterogéneos

En este caso se utilizan tres robots donde sólo uno puede realizar cualquier tarea requerida por la misión, la definición para este caso es la siguiente:

- El conjunto de robot cooperativo, $R=\{r_{rojo}, r_{verde}, r_{azul}\}$
- Los comportamientos de alto nivel, $A_{rojo}=\{a_{rojo\ 2}, a_{rojo\ 3}\}$, $A_{verde}=\{a_{verde\ 2}, a_{verde\ 3}\}$ y $A_{azul}=\{a_{azul\ 1}, a_{azul\ 2}, a_{azul\ 3}\}$
- Las funciones de traducción entre comportamientos de alto nivel y tareas, $h_i(a_{i1})=CRI$, $h_i(a_{i2})=CRD$ y $h_i(a_{i3})=CRA$, $i \in \{rojo,verde,azul\}$.

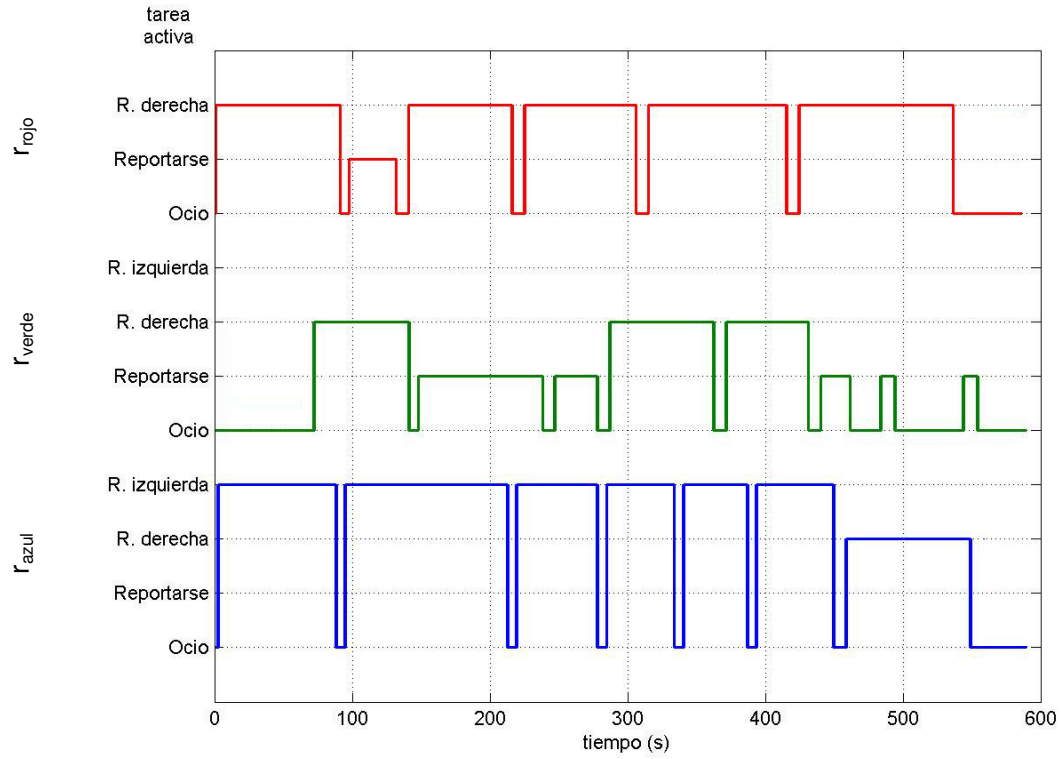


Figura 13. Asignación de tareas utilizando TC/N-ALLIANCE para un equipo heterogéneo.

En la Figura 13 se muestra una asignación de tareas a robots realizada cuando todo el equipo se rige por las reglas de control de la estrategia TC/N-ALLIANCE.

El robot rojo elige durante la misión las siguientes tareas CRD, CRA, CRA, CRD, CRD y CRA. El robot verde elige durante la misión las siguientes tareas CRD, CRA, CRI, CRD, CRD, CRD, CRD, CRI, CRD, CRA, CRA y CRA. El robot azul elige durante la misión las siguientes tareas CRI, CRI, CRI, CRI, CRI, CRI y CRD.

El robot azul determina en tiempo 550 que no hay más alimento, el robot verde determina en tiempo 420 y el robot rojo en tiempo 540.

Tabla 7. Resumen de datos para TC/N-ALLIANCE para un equipo heterogéneo

Dato	**Valor**
Cantidad de robots	3
Cantidad de tareas	3
Equipo homogéneo	No
Cantidad de ejecuciones de CRI (#CRI)	6
Cantidad de ejecuciones de CRD (#CRD)	9
Cantidad de ejecuciones de CRA	6
Duración de la misión	550
Duración de la misión/(#CRI+#CRD)	36

Cuando los robots comienzan a ejecutar, el azul elige la tarea recolectar izquierda, el verde la tarea recolectar derecha y el rojo queda ocioso permitiendo que sus

compañeros realicen la tarea que eligieron. Como la estrategia utilizada es TC/N-ALLIANCE era de esperar que el robot azul comenzara a ejecutar la recolección por la izquierda.

En la Tabla 7 se incluye un resumen de los datos más representativos de la misión que permiten sacar conclusiones de rendimiento respecto de otras estrategias.

6.3.2. L/N-ALLIANCE

Para esta estrategia se presenta primero una misión con tres robots heterogéneos y luego una misión con seis robots homogéneos.

6.3.2.1 Misión con tres robots heterogéneos

En este caso se utilizan tres robots pero no todos los robots pueden ejecutar cualquier tarea requerida por la misión. La definición para este caso es la siguiente:

- El conjunto de robot cooperativo, $R=\{r_{rojo}, r_{verde}, r_{azul}\}$
- Los comportamientos de alto nivel, $A_{rojo}=\{a_{rojo\ 1}, a_{rojo\ 2}, a_{rojo\ 3}\}$, $A_{verde}=\{a_{verde\ 1}, a_{verde\ 3}\}$ y $A_{azul}=\{a_{azul\ 1}, a_{azul\ 3}\}$
- Las funciones de traducción entre comportamientos de alto nivel y tareas, $h_i(a_{i1})=CRI$, $h_i(a_{rojo2})=CRD$ y $h_i(a_{i3})=CRA$, $i \in \{rojo,verde,azul\}$.

En la Figura 14 se muestra la asignación de tareas a un equipo robótico heterogéneo controlado por la estrategia L/N-ALLIANCE donde sólo el robot rojo puede ejecutar la tarea recolectar derecha.

Como se ve en la figura, al no tener en cuenta el task coverage, el robot rojo elige la tarea RecolectarIzquierda que es compartida por todos los robots dejando al resto de los compañeros de equipo ociosos mientras él ejecuta esta tarea. Si en cambio se usara la estrategia TC/N-ALLIANCE el robot rojo elegiría la tarea RecolectarDerecha permitiéndole a uno de sus compañeros ejecutar la tarea RecolectarIzquierda y mejorando de esta forma el rendimiento del equipo. Si se observa el tiempo total requerido para realizar la misión es de 600 segundos, en caso de utilizar la estrategia TC/N-ALLIANCE se hubiera obtenido un tiempo aproximado de 540 segundos. Esto se obtiene permitiendo ejecutar al robot verde o azul la tarea de recolección por al derecha, y desplazando a la izquierda la gráfica de asignación de tareas para el robot rojo. Este valor es similar al obtenido en la ejecución de la misión con TCN-ALLIANCE con tres robots heterogéneos.

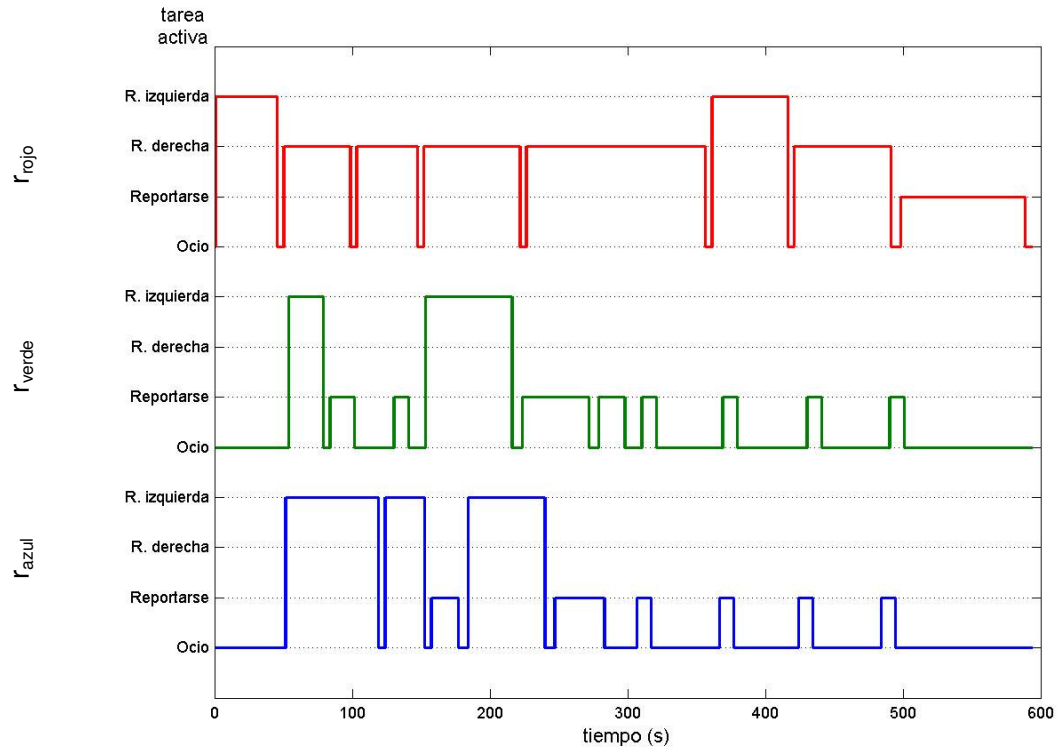


Figura 14. Asignación utilizando L/N-ALLIANCE

Al igual que antes se incluye en la Tabla 8 el resumen de la misión cuando se utiliza la estrategia L/N-ALLIANCE para controlar al equipo robótico. En la misma se observa que aún cuando la misión requiera menos tiempo para ser ejecutada la relación duración de la misión y la cantidad de tareas de recolección realizadas es menor en el caso de TC/N-ALLIANCE, lo cual como se señaló antes es un indicador más adecuado para medir el rendimiento del equipo. En el caso anterior se obtiene un rendimiento un 17% menor.

Tabla 8. Resumen de datos para L/N-ALLIANCE

Dato	**Valor**
Cantidad de robots	3
Cantidad de tareas	3
Equipo homogéneo	No
Cantidad de ejecuciones de CRI (#CRI)	7
Cantidad de ejecuciones de CRD (#CRD)	7
Cantidad de ejecuciones de CRA	14
Duración de la misión	600
Duración de la misión/(#CRI+#CRD)	41

6.3.2.2 Misión con seis robots homogéneos

A continuación se presentan datos obtenidos de la ejecución del caso de estudio utilizando la siguiente configuración:

- El conjunto de robots cooperativo, $R=\{r_{amarillo}, r_{violeta}, r_{celeste}, r_{rojo}, r_{verde}, r_{azul}\}$
- Los comportamientos de alto nivel, $A_{amarillo}=\{a_{amarillo\ 1}, a_{amarillo\ 2}, a_{amarillo\ 3}\}$, $A_{violeta}=\{a_{violeta\ 1}, a_{violeta\ 2}, a_{violeta\ 3}\}$, $A_{celeste}=\{a_{celeste\ 1}, a_{celeste\ 2}, a_{celeste\ 3}\}$, $A_{rojo}=\{a_{rojo\ 1}, a_{rojo\ 2}, a_{rojo\ 3}\}$, $A_{verde}=\{a_{verde\ 1}, a_{verde\ 2}, a_{verde\ 3}\}$, $A_{azul}=\{a_{azul\ 1}, a_{azul\ 2}, a_{azul\ 3}\}$

Las funciones de traducción entre comportamientos de alto nivel y tareas, $h_i(a_{i1})$=CRI, $h_i(a_{i2})$=CRD y $h_i(a_{i3})$=CRA, $i \in$ {amarillo, violeta, celeste, rojo, verde, azul}.

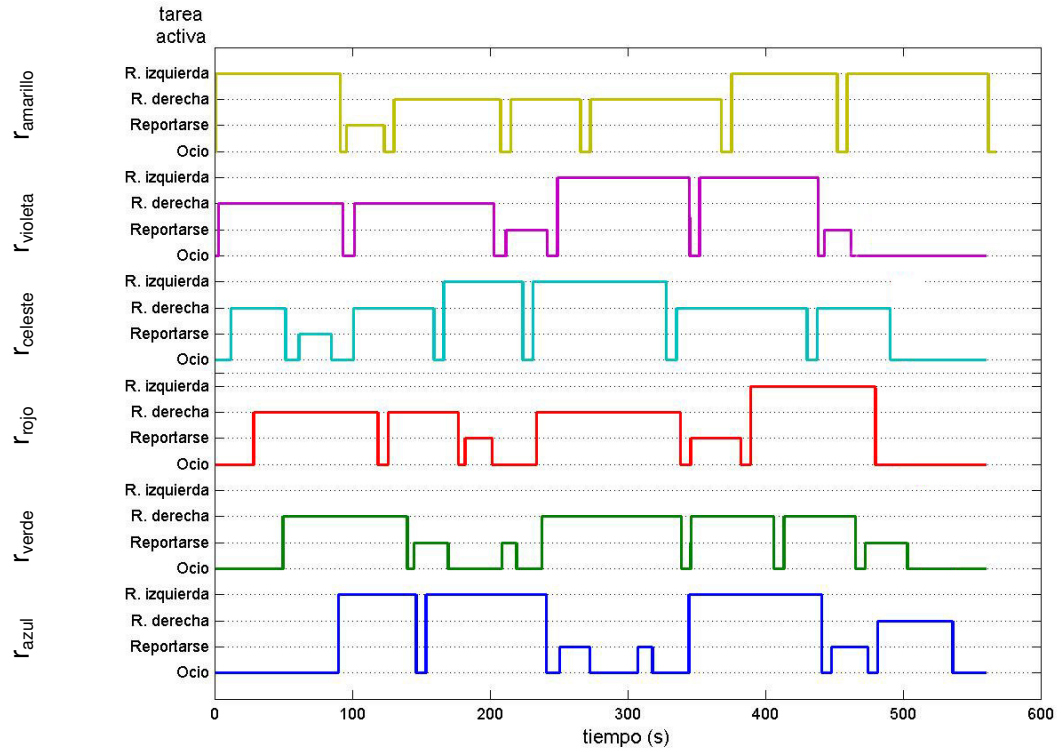


Figura 15. Asignación de tareas para seis robots utilizando L/N-ALLIANCE.

En la Figura 15 se muestra la asignación de tareas obtenida para la configuración antes definida utilizando como estrategia de asignación L/N-ALLIANCE. Al igual que antes se incluye en la Tabla 9 el resumen de la misión cuando se utiliza la estrategia L/N-ALLIANCE para controlar al equipo robótico de seis miembros.

Tabla 9. Resumen de datos para L/N-ALLIANCE para seis robots.

Dato	Valor
Cantidad de robots	6
Cantidad de tareas	3
Equipo homogéneo	Si
Cantidad de ejecuciones de CRI (#CRI)	10
Cantidad de ejecuciones de CRD (#CRD)	17
Cantidad de ejecuciones de CRA	12
Duración de la misión	560
Duración de la misión/(#CRI+#CRD)	21

Además se incluye la Tabla 10 que permite conocer los momentos exactos en donde se producen los cambios de asignación de tareas y la cantidad de robots asignados a cada tarea a medida que evoluciona el tiempo.

Tabla 10. Cantidad de robots asignados a cada tarea usando L/N-ALLIANCE.

	Tiempo	0	1	3	12	29	50	51	61	85	90	91	93	96	101	102	119	123	126	131	140	145	146
Cantidad de robots	Ocio	6	5	5	4	3	2	1	2	1	2	1	2	3	2	1	0	1	2	1	0	1	0
	CRA	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	0	0	0	1
	CRI	0	0	0	1	2	3	4	3	3	3	3	3	2	2	3	4	3	3	4	5	4	4
	CRD	0	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1

(a)

	Tiempo	154	159	167	170	177	182	201	203	208	209	212	215	219	224	231	234	238	241	241	241	249	251
Cantidad de robots	Ocio	1	0	1	0	1	2	1	2	3	4	3	2	1	2	3	2	1	0	1	2	1	0
	CRA	1	1	1	1	0	0	1	0	0	0	1	2	2	1	1	1	1	1	1	0	0	1
	CRI	4	4	3	3	3	2	2	2	1	0	0	0	1	1	1	1	2	3	3	3	3	3
	CRD	0	1	1	2	2	2	2	2	2	2	2	2	2	2	1	2	2	2	1	1	2	2

(b)

	Tiempo	266	272	273	308	318	328	336	338	339	345	345	346	346	353	368	376	382	390	406	414	430	438
Cantidad de robots	Ocio	1	2	1	0	1	2	1	2	3	2	3	2	1	0	1	0	1	0	1	0	0	1
	CRA	1	0	0	1	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0
	CRI	2	2	3	3	3	3	4	3	2	2	2	2	3	3	2	2	2	2	1	2	2	1
	CRD	2	2	2	2	2	1	1	1	1	2	1	1	1	2	2	3	3	4	4	4	4	4

(c)

	Tiempo	438	441	443	448	452	460	462	465	470	473	474	480	482	487	490	498	503	511	536	561
Cantidad de robots	Ocio	0	1	2	1	0	1	0	1	2	2	1	2	3	2	2	3	3	4	4	5
	CRA	0	0	0	1	2	2	2	1	1	1	2	1	1	1	1	1	1	0	0	0
	CRI	2	2	2	2	2	2	2	2	1	1	1	1	1	2	2	1	1	1	1	0
	CRD	4	3	2	2	2	1	2	2	2	2	2	2	1	1	1	1	1	1	1	1

(d)

6.3.3. CR/N-ALLIANCE

Para esta estrategia se presenta primero una misión con tres robots y luego una misión con seis robots, en ambos casos se trabaja con robots homogéneos.

6.3.3.1 Misión con tres robots

En este caso se utiliza la configuración usada para probar la estrategia TC/N-ALLIANCE. El equipo es armado con tres robots donde cada uno puede realizar cualquier tarea requerida por la misión, la definición para este caso es la siguiente:

- El conjunto de robot cooperativo, $R=\{r_{rojo}, r_{verde}, r_{azul}\}$
- Los comportamientos de alto nivel, $A_{rojo}=\{a_{rojo\ 1}, a_{rojo\ 2}, a_{rojo\ 3}\}$, $A_{verde}=\{a_{verde\ 1}, a_{verde\ 2}, a_{verde\ 3}\}$, $A_{azul}=\{a_{azul\ 1}, a_{azul\ 2}, a_{azul\ 3}\}$

Las funciones de traducción entre comportamientos de alto nivel y tareas, $h_i(a_{i1})=CRI$, $h_i(a_{i2})=CRD$ y $h_i(a_{i3})=CRA$, $i \in \{rojo,verde,azul\}$.

Como se puede apreciar en la Figura 16, todos los robots ni bien comienza la misión eligen una tarea para ejecutar tratando de distribuirse las tareas.

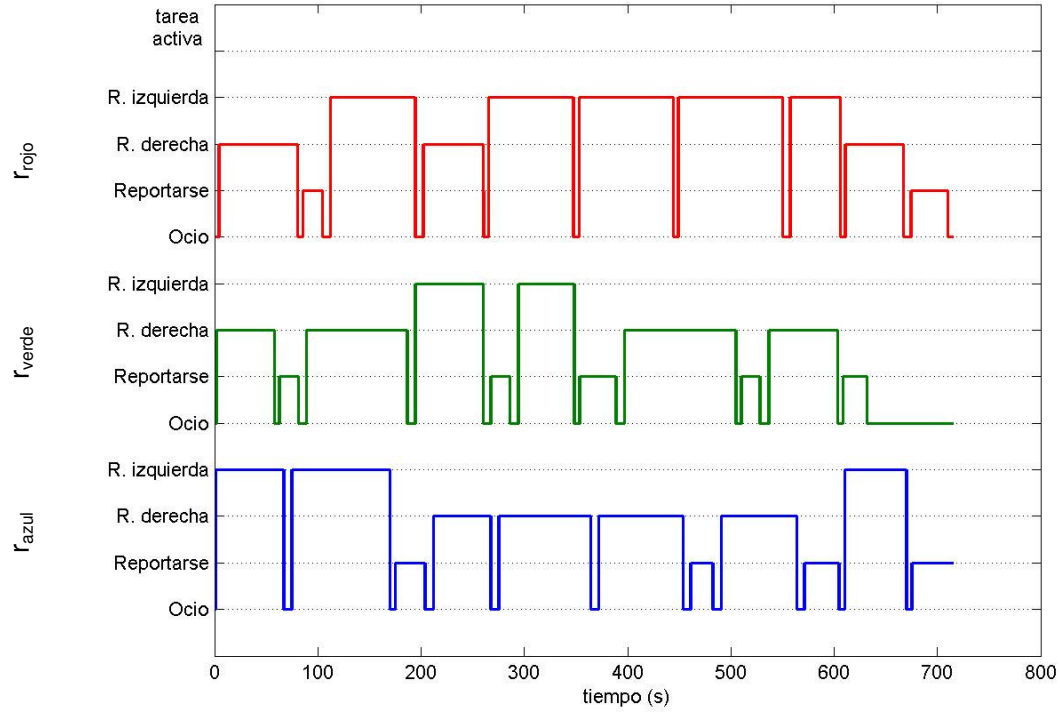


Figura 16. Asignación de tareas utilizando CR/N-ALLIANCE

Como al comienzo de la misión no es necesario realizar un reporte de actividad la información sensorial asociada a este comportamiento indica que no es necesario activarlo, esto lleva a que dos robots elijan una misma tarea de recolección para ejecutar, en este caso ocurre que tanto el robot rojo como el verde optan por ejecutar la tarea RecolectarDerecha mientras el robot azul opta por la tarea RecolectarIzquierda.

Tabla 11. Resumen de datos para CR/N-ALLIANCE

Dato	Valor
Cantidad de robots	3
Cantidad de tareas	3
Equipo homogéneo	Si
Cantidad de ejecuciones de CRI (#CRI)	10
Cantidad de ejecuciones de CRD (#CRD)	11
Cantidad de ejecuciones de CRA	8
Duración de la misión	680
Duración de la misión/(#CRI+#CRD)	31.9

Respecto a la distribución de tareas se puede apreciar que los robots estuvieron trabajando sobre distintas tareas en tiempo 80, 100, 180, 280, 350, 460, 520, 580 y 620. El tiempo consumido por la misión fue de 680 segundos. En la

Tabla 11 se resumen las características y resultados obtenidos al ejecutar el caso de estudio utilizando CR/N-ALLIANCE como estrategia de control para los robots. Se puede apreciar un rendimiento superior en un 6% si se compara esta estrategia con L/N-ALLIANCE.

6.3.3.2 Misión con seis robots

El equipo es armado con seis robots donde cada uno puede realizar cualquier tarea requerida por la misión, la definición para este caso es la siguiente:

- El conjunto de robots cooperativo, $R=\{r_{amarillo}, r_{violeta}, r_{celeste}, r_{rojo}, r_{verde}, r_{azul}\}$.
- Los comportamientos de alto nivel, $A_{amarillo}=\{a_{amarillo\ 1}, a_{amarillo\ 2}, a_{amarillo\ 3}\}$, $A_{violeta}=\{a_{violeta\ 1}, a_{violeta\ 2}, a_{violeta\ 3}\}$, $A_{celeste}=\{a_{celeste\ 1}, a_{celeste\ 2}, a_{celeste\ 3}\}$, $A_{rojo}=\{a_{rojo\ 1}, a_{rojo\ 2}, a_{rojo\ 3}\}$, $A_{verde}=\{a_{verde\ 1}, a_{verde\ 2}, a_{verde\ 3}\}$, $A_{azul}=\{a_{azul\ 1}, a_{azul\ 2}, a_{azul\ 3}\}$

Las funciones de traducción entre comportamientos de alto nivel y tareas, $h_i(a_{i1})=CRI$, $h_i(a_{i2})=CRD$ y $h_i(a_{i3})=CRA$, $i \in \{amarillo, violeta, celeste, rojo, verde, azul\}$.

En la Figura 17 se muestra la asignación de tareas cuando se utiliza un equipo robótico formado por seis robots homogéneos.

Si se compara la Figura 15 y la Figura 17, aun cuando estas difieran levemente en la escala de tiempo, se observa un decremento en el tiempo que permanecen ociosos los robots.

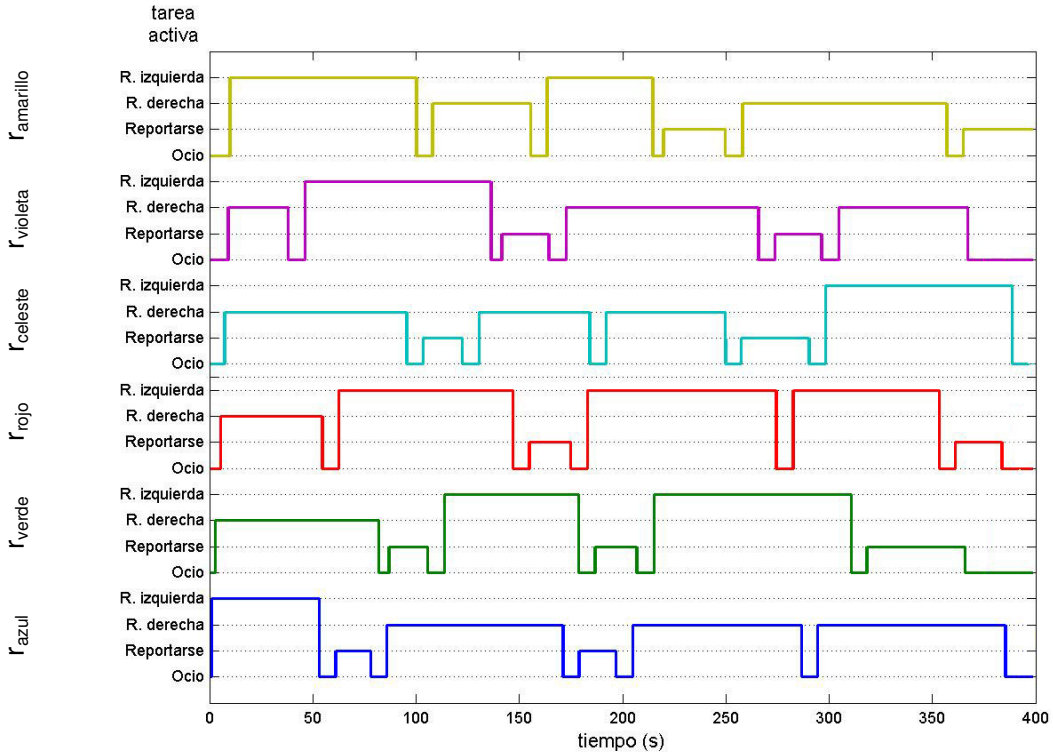


Figura 17. Asignación de tareas para seis robots utilizando CR/N-ALLIANCE.

En este caso se puede notar un decremento importante en el tiempo de ejecución logrado principalmente por una buena distribución de tareas, lo cual redunda en una menor interferencia entre los robots. En la Tabla 12 se muestra el resumen de la misión, en la misma se puede apreciar el valor de la relación duración de la misión y cantidad de tareas de recolección ejecutadas, presentando el mejor valor obtenido si se compara con el resto de las tablas.

Tabla 12. Resumen de datos para CR/N-ALLIANCE para seis robots.

Dato	**Valor**
Cantidad de robots	6
Cantidad de tareas	3
Equipo homogéneo	Si
Cantidad de ejecuciones de CRI (#CRI)	10
Cantidad de ejecuciones de CRD (#CRD)	12
Cantidad de ejecuciones de CRA	12
Duración de la misión	380
Duración de la misión/(#CRI+#CRD)	17

En la Tabla 13 se permite conocer los momentos exactos en donde se producen los cambios de asignación de tareas y la cantidad de robots asignados a cada tarea a medida que evoluciona el tiempo. Si se compara esta tabla con la Tabla 10 se puede observar que el algoritmo CR/N-ALLIANCE logra una distribución más uniforme.

Tabla 13. Cantidad de robots asignados a cada tarea usando CR/N-ALLIANCE.

	Tiempo	1	3	5	7	9	10	38	46	53	54	61	63	78	82	86	87	95	100	103	106	108	114
Cantidad de robots	**Ocio**	6	5	4	3	2	1	0	1	0	1	2	1	0	1	2	1	0	1	2	1	2	1
	CRA	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	2	1	1
	CRI	0	0	1	2	3	4	4	3	3	3	2	2	2	2	1	2	2	1	1	1	1	2
	CRD	0	1	1	1	1	1	2	2	3	2	2	2	3	3	3	3	3	3	2	2	2	2

(a)

	Tiempo	122	131	136	142	147	155	155	164	164	165	171	173	175	179	179	183	184	187	192	197	205	207
Cantidad de robots	**Ocio**	0	1	0	1	0	1	0	1	0	1	2	1	2	3	2	1	2	2	1	0	1	0
	CRA	1	0	0	0	1	1	2	2	2	1	1	1	0	0	1	1	1	1	2	2	1	1
	CRI	2	2	3	3	3	3	3	2	2	2	1	2	2	2	2	2	1	1	1	2	2	3
	CRD	3	3	3	2	2	1	1	1	2	2	2	2	2	1	1	2	2	2	2	2	2	2

(b)

	Tiempo	215	215	220	250	250	258	258	266	274	274	283	287	290	295	296	298	305	311	318	353	357	361
Cantidad de robots	**Ocio**	1	2	1	0	1	2	1	0	1	0	1	0	1	2	1	2	1	0	1	0	1	2
	CRA	0	0	0	1	0	0	1	1	1	2	2	2	2	1	1	0	0	0	0	1	1	1
	CRI	3	3	3	3	3	2	2	3	2	2	2	2	1	1	2	2	2	3	3	3	3	2
	CRD	2	1	2	2	2	2	2	2	2	2	1	2	2	2	2	2	3	3	2	2	1	1

(c)

6.3.4. Análisis de Resultados

En la sección anterior se presentaron los resultados obtenidos al ejecutar las distintas estrategias de control, si bien se presentó un análisis preliminar y simple de dichos resultados, en este punto se pretende comparar de manera objetiva dichos resultados.

La estrategia TC/N-ALLIANCE demuestra una mejor eficiencia que L/N-ALLIANCE en los resultados presentados, esto se aprecia al comparar las Tabla 7 y Tabla 8, la instancia del problema que se presenta trata justamente de marcar el problema que se detectó en la estrategia de asignación de L-ALLIANCE.

Luego se pretendió comparar la estrategia CR/N-ALLIANCE con L/N-ALLIANCE lo cual en este caso redundó en un mejor rendimiento del equipo al realizar la misión, debido principalmente a la realización de la misma tarea en paralelo por varios robots con una baja interferencia entre los mismos. Esta afirmación se obtiene al analizar desde la Tabla 8 a la Tabla 13. Como era de esperar la estrategia CR/N-ALLIANCE logra una mejor distribución lo que lleva a una menor interferencia entre los robots logrando menores tiempos de ejecución por tarea. L/N-ALLIANCE no puede garantizar una asignación uniforme pues su mecanismo automático de actualización no se plantea en ningún momento lograr ese objetivo. En la Tabla 10 se puede apreciar la asignación del equipo, lo cual es casual pues al no garantizar la distribución uniforme podrían obtenerse ejecuciones de la misión donde siempre haya un robot ejecutando la tarea de recolección a la izquierda y cinco ejecutando la tarea de recolección a la derecha lo cual genera una fuerte interferencia entre los robots de la derecha

obteniendo tiempos de ejecución muy por encima de los obtenidos utilizando la estrategia CR/N-ALLIANCE.

6.4. Otros experimentos

En la sección anterior se presentaron ejecuciones particulares de las distintas estrategias utilizando el caso de estudio propuesto por Parker, en todos los casos se utilizó el simulador YAKS. En esta sección se presentan misiones virtuales generadas variando la cantidad de robots que participan, la cantidad de tareas, tiempo requerido para ejecutar cada tarea, el task coverage y la cantidad de repeticiones. Una vez establecidos los parámetros de la misión y la estrategia a utilizar, se ejecuta un proceso para cada robot, el cual instancia la estrategia seleccionada. En este caso la ejecución de una tarea se implementa esperando un cierto tiempo, el cual está determinado por el tiempo requerido para la ejecución de la misma.

Para esto se generaron cerca de 2000 misiones virtuales, las mismas se generaron variando la cantidad de tareas entre 2 y 20, la cantidad de robots entre 2 y 20, y el task coverage entre 1 y la cantidad de robots de cada caso de prueba.

Para generar las misiones virtuales se utilizó el Algoritmo 1.

6.4.1. L/N-ALLIANCE versus TC/N-ALLIANCE

En esta sección se compara el rendimiento de las estrategias L/N-ALLIANCE y TC/N-ALLIANCE, tomando en cuenta los sólo estas dos estrategias.

```
generador(in int maxRobots, in int maxTareas, in int maxRepes,
 in int minTaskTime, in int maxTaskTime)
 int repes []; int tt[];
begin
 for iterRobots=2 to maxRobots
  for iterTareas=2 to maxTareas
    repes := sortearRepeticiones() // para cada tarea sortear la cantidad
    // de repeticiones entre 1 y maxRepes
    tt := sortearTaskTime(); // para cada par (robot,tarea) sortea el task
    // time entre minTaskTime y maxTaskTime
    ejecutarMisionLN-ALLIANCE(iterRobots, iterTareas, repes, tt);
    // ejecuta iterRobots procesos que ejecutan la estrategia LN-ALLIANCE
    guardarInfo(); // almacena en disco toda la información de la misión
    ejecutarMisionTCN-ALLIANCE(iterRobots, iterTareas, repes, tt);
    guardarInfo();
    ejecutarMisionCRN-ALLIANCE(iterRobots, iterTareas, repes, tt);
    guardarInfo();
  end
 end
```

Algoritmo 1. Generador de misiones virtuales.

De la ejecución de estas misiones virtuales se desprende que el 33% de los casos de prueba la estrategia TC/N-ALLIANCE obtuvo un mejor rendimiento. En la Figura 18 se grafica la diferencia entre el tiempo de ejecución de la misión utilizando la estrategia TC/N-ALLIANCE y el tiempo de ejecución de la misión utilizando la estrategia L/N-ALLIANCE.

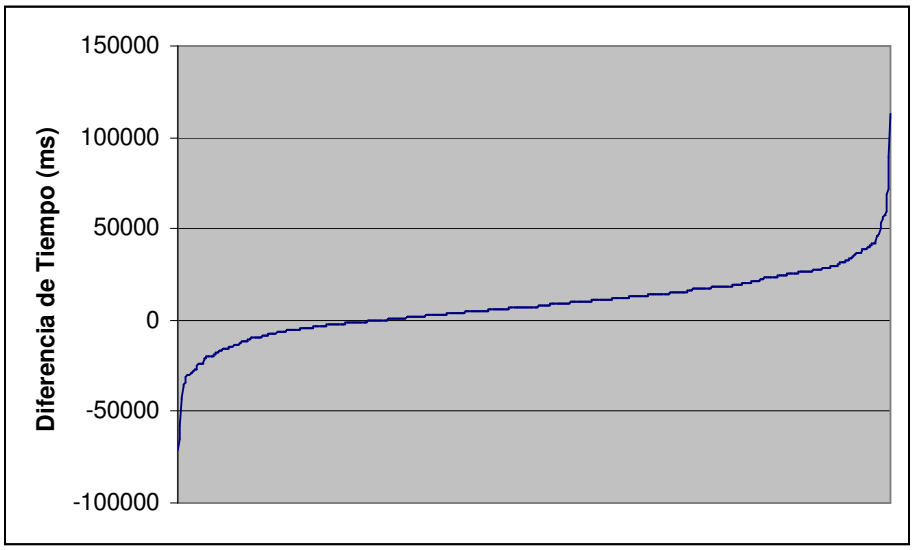


Figura 18. Gráfica tiempo de ejecución de TC/N-ALLIANCE menos el tiempo de ejecución de L/N-ALLIANCE para las misiones virtuales.

En la Tabla 14 se presentan los valores medio, máximo y mínimo de la diferencia entre el tiempo de ejecución de la misión utilizando la estrategia TC/N-ALLIANCE y el tiempo de ejecución de la misión utilizando la estrategia L/N-ALLIANCE.

Tabla 14. Datos comparativos de los tiempos de ejecución.

	Diferencia Promedio (ms)	Máxima Diferencia (ms)	Mínima Diferencia (ms)
TE(TC)-TE(L)	8304	112821	-71332
TE(CR)-TE(L)	-5545	145589	-82240
TE(CR)-TE(TC)	13850	113090	-79186

6.4.2. L/N-ALLIANCE versus CR/N-ALLIANCE

En esta sección se compara el rendimiento de las estrategias L/N-ALLIANCE y CR/N-ALLIANCE.

De estos resultados se desprende que el 73% de los casos de prueba la estrategia CR/N-ALLIANCE obtuvo un mejor rendimiento.

En la se Figura 19 grafica la diferencia entre el tiempo de ejecución de la misión utilizando la estrategia CR/N-ALLIANCE y el tiempo de ejecución de la misión utilizando la estrategia L/N-ALLIANCE.

Al igual que antes, en la Tabla 14 se presentan los valores medio, máximo y mínimo de la diferencia entre el tiempo de ejecución de la misión utilizando la estrategia CR/N-ALLIANCE y el tiempo de ejecución de la misión utilizando la estrategia L/N-ALLIANCE.

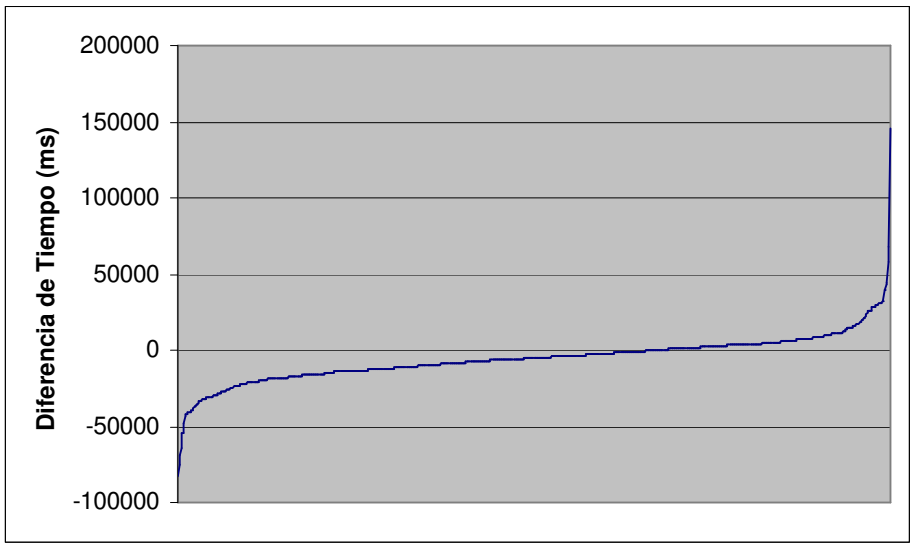


Figura 19. Gráfica tiempo de ejecución de CR/N-ALLIANCE menos el tiempo de ejecución de L/N-ALLIANCE para las misiones virtuales.

6.4.3. TC/N-ALLIANCE versus CR/N-ALLIANCE

En esta sección se compara el rendimiento de las estrategias T/N-ALLIANCE y CR/N-ALLIANCE.

De estos resultados se desprende que el 87% de los casos de prueba la estrategia CR/N-ALLIANCE obtuvo un mejor rendimiento.

En la Figura 20 se grafica la diferencia entre el tiempo de ejecución de la misión utilizando la estrategia CR/N-ALLIANCE y el tiempo de ejecución de la misión utilizando la estrategia TC/N-ALLIANCE.

Al igual que antes, en la Tabla 14 se presentan los valores medio, máximo y mínimo de la diferencia entre el tiempo de ejecución de la misión utilizando la estrategia CR/N-ALLIANCE y el tiempo de ejecución de la misión utilizando la estrategia TC/N-ALLIANCE.

Si se comparan simultáneamente las tres estrategias para las misiones virtuales se obtiene que en el 63% de las misiones la estrategia CR/N-ALLIANCE obtiene un mejor rendimiento, en el 30% de las misiones la estrategia L/N-ALLIANCE obtiene un mejor rendimiento y en el 7% restante la estrategia TC/N-ALLIANCE se desempeña mejor.

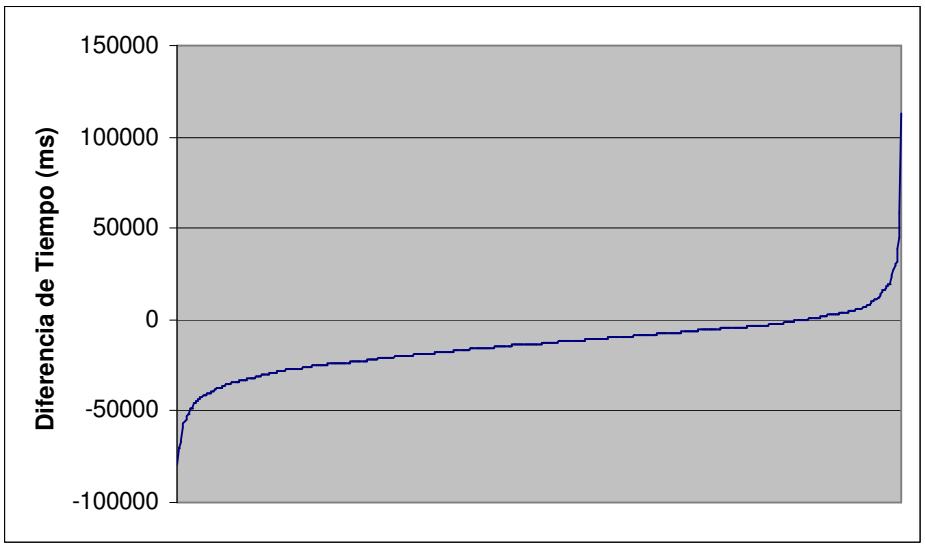


Figura 20. Gráfica tiempo de ejecución de CR/N-ALLIANCE menos el tiempo de ejecución de TC/N-ALLIANCE para las misiones virtuales.

7. Diseño de los comportamientos

7.1. Introducción

En este capítulo se presentan los detalles de las tareas de recolección y reporte de actividad indicadas en el Capítulo 6. Las mismas son utilizadas en dicho capítulo para definir la misión a realizar por el equipo robótico. Estas tareas o comportamientos de alto nivel se construyen a partir de comportamientos reactivos.

Las tareas que requiere la misión deben permitir al robot recolectar objetos y reportar la actividad actual, estos comportamientos de alto nivel se describen aquí como máquinas de estado, donde los estados representan comportamientos reactivos y las transiciones se realizan en función de la percepción del robot.

A continuación se detalla la tarea de recolección de objetos y luego la tarea de reporte de actividad.

7.2. Recolectar Objetos

La tarea de recolectar objetos coloca al robot en un entorno junto con varios objetos que debe encontrar y transportar hacia un determinado lugar. Este problema es estudiado en varios trabajos basándose en el comportamiento que ciertos insectos muestran al encontrar comida y transportarla a su nido u hogar (home) [Kube1992, Kube1993, Mataric1994, Mataric1995, Mataric1997, Mataric1998, Werger1999]. También se ha estudiado por su similitud con tareas de recolección de rocas en planetas desconocidos [Brooks1989] o aplicaciones de limpieza [Parker1998, Jung1999].

En los trabajos antes indicados se utilizan distintos sensores para resolver el problema. En algunos trabajos el home está determinado por una posición dentro del entorno mientras que en otros por una fuente emisora de luz y el robot dispone de sensores de luz para transportar el alimento al home. Otra diferencia es respecto a la ubicación de la fuente de alimento. En algunos trabajos la fuente es fija y está dada por su posición dentro del entorno mientras que en otros los robots deben encontrar por si mismos el alimento dentro del entorno. Además de estas diferencias entre las capacidades de percepción se observaron también diferencias en el tipo de actuadores utilizados. En el caso más simple el robot dispone de una pinza que le permite agarrar el alimento y trasportarlo hasta el nido, otros al no disponer de pinzas deben empujar el objeto con su cuerpo hasta el nido.

El sistema básico que acompaña al pequeño robot Khepera no dispone de pinzas, aún cuando estas pueden ser adquiridas por separado e integradas al mismo de manera muy simple pero a un costo elevado. Por este motivo se decidió resolver el problema utilizando únicamente las características del equipo básico Khepera, asumiendo las dificultades que esto implica.

El comportamiento de recolectar objetos se implementó a partir de varios comportamientos reactivos muy simples. Estos comportamientos simples se implementaron utilizando reglas que evalúan los sensores y determinan la potencia de los actuadores. Se experimentó utilizando aprendizaje por refuerzo [Kaelbling1996, Sutton1998], y redes neuronales [Dayhoff1990, Haykin1994, Hertz1991, Moody1989] de forma de aprender los comportamientos simples utilizando las propuestas [Santos1999] y [Touzet1997] pero no se obtuvieron buenos resultados.

7.2.1. Implementación

El comportamiento que recolecta objetos se implementó como una máquina de estados, donde el estado indica el comportamiento activo y las transiciones se indican como predicados sobre los sensores. El comportamiento activo es elegido para interactuar con el entorno. En la Figura 21 se muestra la máquina de estados diseñada, en el contexto de este trabajo, para implementar el comportamiento recolectar objetos. En la misma se aprecian los comportamientos simples IrPorComida, ExplorarZona, RodearHorario, RodearAntiHorarios, MirarAlHome, Empujar y EvitarObstaculos, y los predicados veoAlgo, enZonaComida, cercaDelHome, fueraDeRumbo, homeALaIzquierda, comidaLaIzquierda, comidaLaDerecha, miroAlHome y perpendicularAlHome.

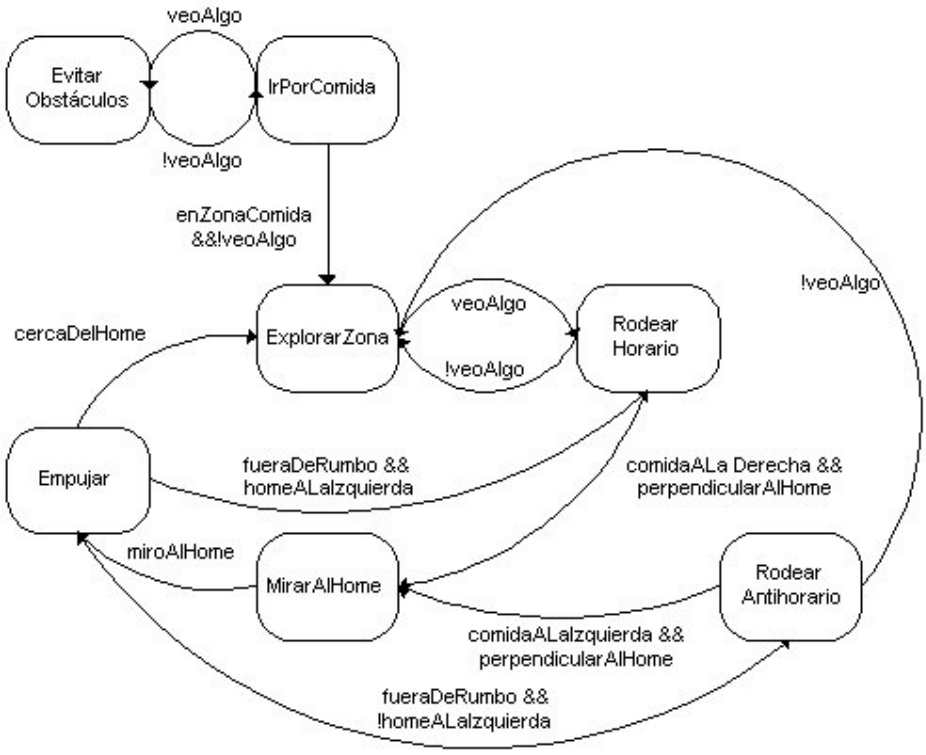


Figura 21. Máquina de estados para el comportamiento recolectar objetos.

El comportamiento recolectar objetos obtiene de los sensores la siguiente información:

- valor de sensores infrarrojos del robot Khepera,
- posición y dirección del robot Khepera en el mundo,
- posición de los objetos (centro de la zona de recolección) y
- posición del home,

y debe controlar en todo momento los dos motores del robot.

7.2.1.1 Los predicados sensoriales

Los predicados determinan bajo que situación se realiza una transición entre estados o lo que es lo mismo se pasa a ejecutar otro comportamiento. Estos predicados se definen como expresiones booleanas en función de los valores de los sensores. Para el comportamiento recolectar objetos es necesario disponer de la lectura de los sensores infrarrojos del robot, la posición del robot y su dirección, la posición del centro de la zona de recolección y la posición del home.

Para obtener la lectura de los sensores infrarrojos del robot se utiliza la notación sensoresKhepera(i), i∈{0,...,7}. Los sensores de proximidad se ubican físicamente en la superficie del robot según se muestra en la superficie del robot según se muestra en la Figura 22.

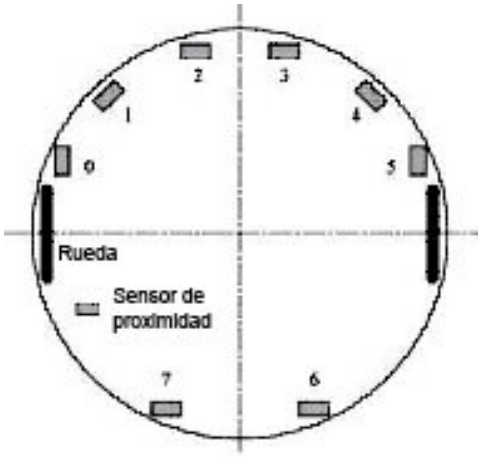


Figura 22. Ubicación de los sensores en el robot Khepera.

Predicado veoAlgo

Este predicado devuelve verdadero si el robot determina que está cerca de algún objeto. Para esto alcanza con utilizar únicamente los sensores infrarrojos del robot. Cuando se trabaja con robots reales los sensores normalmente tienen ruido por lo que se define un valor constante que determina valor tolerable de ruido (UMBRAL_RUIDO) y en caso de que algún sensor (*sensorKhepera*(i)) muestre una lectura mayor a dicho umbral se concluye que se está cerca de un objeto o se ve algo. Formalmente este predicado se define de la siguiente manera:

$$veoAlgo(sensoresKhepera) = \begin{cases} true & \exists i.\left(sensoresKhepera(i) > UMBRAL\_RUIDO\right) \\ false & en\ otro\ caso \end{cases}.$$

Predicado enZonaComida

Este predicado devuelve verdadero si el robot se encuentra cerca de la comida, para lo cual es necesario conocer la posición del robot (*posRobot*) y la posición de la comida (*posAlimento*). Para definir este predicado se considera un valor constante (CERCA_ALIMENTO) a partir del cual se puede decir que el robot está cerca del alimento. Formalmente este predicado se define de la siguiente manera:

$$enZonaAlimento(posRobot, posAlimento) = (distancia(posRobot, posAlimento) < CERCA\_ALIMENTO).$$

La función *distancia* devuelve la distancia entre dos puntos, en este caso los puntos *posRobot* y *posAlimento*, esto es la distancia entre la ubicación del robot y el home.

Predicado cercaDelHome

Este predicado devuelve verdadero si el robot se encuentra cerca del home, para lo cual es necesario conocer la posición del robot (*posRobot*) y la posición del home (*posHome*). Para definir este predicado se considera un valor constante (CERCA_HOME) a partir del cual se puede decir que el robot está cerca del home. Formalmente se define este predicado de la siguiente manera:

$$cercaDelHome(posRobot, posHome) = (distancia(posRobot, posHome) < CERCA\_HOME)\,.$$

Predicado fueraDeRumbo

El predicado *fueraDeRumbo* permite determinar cuando el robot no está posicionado adecuadamente, impidiendo empujar eficientemente el alimento al home. Se define primero el predicado *alineado*, que determina cuando el ángulo entre un punto y la dirección del robot es mayor a un determinado valor (DIF_ANGULO), y a partir de este el predicado *fueraDeRumbo*. Para definir el predicado *alineado* es necesario conocer la posición (*posRobot*) y dirección del robot (*dirRobot*), y la posición del punto con el cual deseo saber si estoy alineado (*puntoAlinear*). Formalmente este predicado se define de la siguiente manera:

$$alineado(puntoAlinear, posRobot, dirRobot) = (anguloEntre(puntoAlinear, posRobot, dirRobot) < DIF\_ANGULO) \wedge \cdot (anguloEntre(puntoAlinear, posRobot, dirRobot) > -DIF\_ANGULO)$$

La función *anguloEntre* devuelve el ángulo entre el robot y el home en radianes como un valor perteneciente a [-π,π].

Luego se define *fueraDeRumbo* en función del predicado *alineado* de la siguiente manera:

$$fueraDeRumbo(posHome, posRobot, dirRobot) = \neg alineado(posHome, posRobot, dirRobot)\,.$$

Predicado homeALaIzquierda

Este predicado determina cuando para el robot el home está a su izquierda. Para definir este predicado es necesario conocer la posición (*posRobot*) y dirección del robot (*dirRobot*), y la posición del home (*posHome*). Formalmente este predicado se define de la siguiente manera:

$$homeALaIzquierda(posHome, posRobot, dirRobot) = (anguloEntre(posHome, posRobot, dirRobot) < 0)\,.$$

Predicado comidaALaIzquierda

Este predicado determina cuando para el robot hay un objeto a recolectar a su izquierda. Para definir este predicado es necesario conocer la lectura de los sensores infrarrojos del robot (*sensorsKhepera*). Formalmente este predicado se define de la siguiente manera:

$$comidaALaIzquierda(sensoresKhepera) = (sensoresKhepera(0) > UMBRAL\_RUIDO)\,.$$

Predicado comidaALaDerecha

Este predicado determina cuando para el robot hay un objeto a recolectar a su derecha. Para definir este predicado es necesario conocer la lectura de los sensores infrarrojos del robot (*sensorsKhepera*). Formalmente este predicado se define de la siguiente manera:

$$comidaALaIzquierda(sensoresKhepera) = (sensoresKhepera(5) > UMBRAL\_RUIDO).$$

Predicado miroAlHome

Este predicado determina cuando la dirección entre el home y el robot es mayor a un determinado valor (ANGULO_MIRAR), lo cual indica que el robot está posicionado adecuadamente permitiendo comenzar a empujar el alimento al home. Para definir este predicado es necesario conocer la posición (*posRobot*) y dirección del robot (*dirRobot*), y la posición del home (*posHome*). Formalmente este predicado se define de la siguiente manera:

$$\begin{aligned} &miroAlHome(posHome, posRobot, dirRobot) = \\ &\qquad (anguloEntre(posHome, posRobot, dirRobot) < \text{ANGULO\_MIRAR}) \wedge . \\ &\qquad (anguloEntre(posHome, posRobot, dirRobot) > -\text{ANGULO\_MIRAR}) \end{aligned}$$

Predicado perpendicularAlHome

Este predicado determina cuando el robot se encuentra posicionado perpendicular al home a menos de un error (ERROR_ANGULO). Para definir este predicado es necesario conocer la posición (*posRobot*) y dirección del robot (*dirRobot*), y la posición del home (*posHome*). Formalmente este predicado se define de la siguiente manera:

$$\begin{aligned} &perpendicularAlHome(posHome, posRobot, dirRobot) = \\ &\left[\begin{array}{l} \left(anguloEntre(posHome, posRobot, dirRobot) < \left(\frac{\pi}{2} + ERROR\_ANGULO\right)\right) \wedge \\ \left(anguloEntre(posHome, posRobot, dirRobot) > \left(\frac{\pi}{2} - ERROR\_ANGULO\right)\right) \end{array}\right] \vee \\ &\left[\begin{array}{l} \left(anguloEntre(posHome, posRobot, dirRobot) < \left(-\frac{\pi}{2} + ERROR\_ANGULO\right)\right) \wedge \\ \left(anguloEntre(posHome, posRobot, dirRobot) > \left(-\frac{\pi}{2} - ERROR\_ANGULO\right)\right) \end{array}\right] \end{aligned}$$

7.2.1.2 Los comportamientos

El comportamiento recolectar objetos se compone de varios comportamientos reactivos que se detallan a continuación. Los comportamientos reactivos se basan en las ideas propuestas en [Braitenberg1985].

Comportamiento EvitarObstaculos

Este comportamiento se encarga de evitar obstáculos cuando el robot se desplaza en busca de alimento. Para la implementación de este comportamiento se evaluaron las propuestas que aparecen en [Touzet1997] y la propuesta de K-Team, optándose por razones de eficiencia por la propuesta de K-Team, al comparar empíricamente la velocidad alcanzada por el robot.

```
darAccion(in double sensoresIR[8], out double actuadores[2])
 const double pesos[2][8]
begin
 for iterAct=0 to 1
  actuadores[iterAct]=pesos[iterAct][0]
  for iterSens=0 to 7 do
   actuadores[iterAct]=actuadores[0]+
   sensores[iterSens]*pesos[iterAct][iterSens+1]
  end
 end
end
```

Algoritmo 2. Comportamiento para evitar obstáculos.

Este comportamiento recibe los valores de los sensores infrarrojos del robot Khepera y devuelve el valor que deben tomar los actuadores del robot, esto es, la velocidad de la rueda izquierda y derecha. Los valores constantes (*pesos*) se ajustan de manera que el robot gire a la derecha si hay un obstáculo a la izquierda y gire a la izquierda si hay un obstáculo a la derecha. El Algoritmo 2 define al comportamiento evitar obstáculos, los valores adecuados para la constante *pesos* pueden obtenerse de la implementación realizada por K-Team para Matlab.

Comportamiento MirarA

Este comportamiento es el encargado de modificar la dirección del robot girando sobre sí mismo hasta que el robot quede mirando un determinado punto del entorno. Para la implementación de este comportamiento es necesario conocer la dirección y posición actual del robot Khepera, y la posición hacia donde se quiere mirar. El Algoritmo 3 define el comportamiento utilizado para mirar hacia un determinado lugar. La función anguloEntre devuelve el ángulo entre el vector dirección del robot y el punto hacia el cual se quiere mirar, si es este ángulo es mayor a la constante de error ERR_ANGULO se devuelve la velocidad de los motores de forma que el robot gire sobre si mismo en sentido horario, si la constante VEL_GIRO es mayor a cero, y cuando el ángulo es menor a la constante de error (el robot está mirando su objetivo) se ajustan los actuadores adecuadamente para detener el movimiento del robot.

```
darAccion(in double posRobot[2],in double dirRobot,
 in posMirar, out double actuadores[2])
 const double ERR_ANGULO, VEL_GIRO, VEL_DETENIDO
begin
 if abs(anguloEntre(posMirar,posRobot,dirRobot))>ERR_ANGULO
   actuadores[0]=VEL_DETENIDO+VEL_GIRO
   actuadores[1]=VEL_DETENIDO-VEL_GIRO
 else
   actuadores[0]=VEL_DETENIDO
   actuadores[1]=VEL_DETENIDO
 end
end
```

Algoritmo 3. Comportamiento para mirar hacia.

Comportamiento IrPorComida

Este comportamiento es el encargado de mover el robot desde su posición actual hasta la posición donde se encuentra el alimento. Para la implementación de este comportamiento se conoce la dirección y posición actual del robot Khepera, y la posición donde se encuentra el alimento. Una implementación posible utilizando máquinas de estado es la que se muestra en la Figura 23.

La idea detrás de esta máquina de estados es muy simple, se define basándose en los comportamientos MirarA (presentado más arriba) visto antes y al comportamiento Avanzar que determina el avance en línea recta del robot. Las transiciones entre los dos estados se determinan con el predicado alineado definido antes.

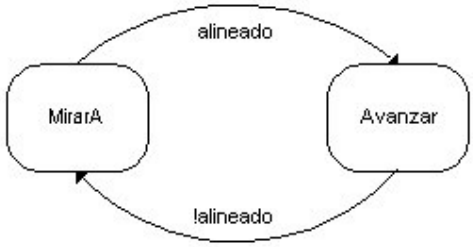
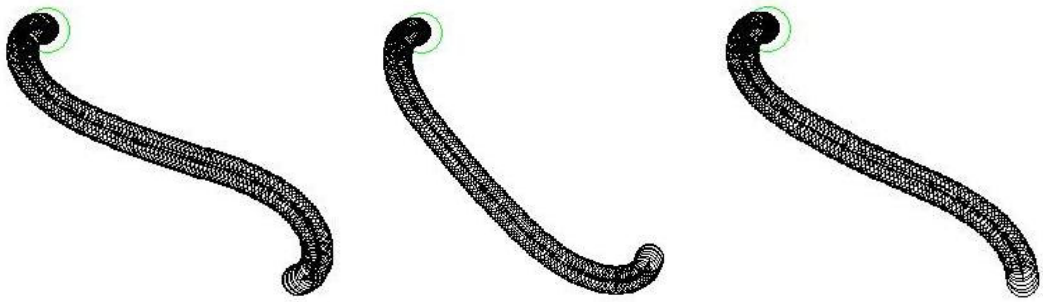


Figura 23. Máquina de estados para el comportamiento IrA.

Esta implementación hace que el movimiento del robot sea tosco pero es muy simple de implementar y entender.

Otra implementación es la propuesta de [Webers2002] basada en el análisis físico del problema. En su artículo se presentan varios algoritmos y en particular el algoritmo denominado Bounded velocity Lyapunov (BV), el cual demostró ser más estable que las otras propuestas. En la Figura 24 se muestra tres imágenes capturadas del simulador YAKS donde se muestra la trayectoria seguida por el robot desde el vértice inferior derecho al vértice superior izquierdo. En la imagen (a) el robot inicia su recorrido con ángulo 0º, en la imagen (b) con ángulo 90º y en la imagen (c) con ángulo 270º, se eligieron estos casos representativos en los cuales se puede apreciar la suavidad de las trayectorias seguidas por el robot. También se verifico la eficiencia del algoritmo comparándolo con la propuesta que utiliza máquina de estados obteniendo mejores resultados el algoritmo BV.



(a) (b) (c)

Figura 24. Trayectorias utilizando el algoritmo BV.

Este último algoritmo fue él utilizado en la implementación del comportamiento de alto nivel recolectar objetos.

Comportamiento ExplorarZona

Este comportamiento mueve al robot según una trayectoria de espiral lo cual le permite recorrer una zona determinada en busca de un objeto para recolectar.

Comportamiento RodearHorario

Este comportamiento es el encargado de mover el robot siguiendo los objetos por su derecha, este comportamiento puede verse, si se observa el robot desde arriba, como que el robot rodea el objeto en sentido horario. En la bibliografía [Santos1999] este comportamiento normalmente se nombra “seguir paredes”. Para la implementación del mismo alcanza con conocer la lectura de los sensores infrarrojos del robot.

El Algoritmo 4 define el comportamiento utilizado para seguir paredes o elementos del mundo utilizando únicamente los sensores infrarrojos del robot.

```
darAccion(in double sensoresIR[8], out double actuadores[2])
 const double VEL_IZQ_DOBLAR_IZQ, VEL_DER_DOBLAR_IZQ,
 VEL_IZQ_DOBLAR_DER, VEL_DER_DOBLAR_DER,
 VEL_IZQ_GIRAR, VEL_DER_GIRAR, VEL_AVANCE
 const double DEBO_ABRIRME, MIN_DEBO_CERRARME,
 MAX_DEBO_CERRARME
begin
 if (sensoresIR[4]>DEBO_ABRIRME)
   actuadores[0] = VEL_IZQ_DOBLAR_IZQ
   actuadores[1] = VEL_DER_DOBLAR_IZQ
 else if (sensoresIR [5]<MAX_DEBO_CERRARME)&&
         (sensoresIR [5]>MIN_DEBO_CERRARME)
   actuadores[0] = VEL_IZQ_DOBLAR_DER
   actuadores[1] = VEL_DER_DOBLAR_DER
 else if (sensoresIR [5]<=MIN_DEBO_CERRARME)
   actuadores[0] = VEL_IZQ_GIRAR
   actuadores[1] = VEL_DER_GIRAR
 else
   actuadores[0] = VEL_AVANCE
   actuadores[1] = VEL_AVANCE
 end
end
```

Algoritmo 4. Comportamiento para seguir paredes por la derecha.

El primer if del algoritmo se utiliza para doblar a la izquierda en las esquinas, el segundo if ayuda a pegarse a la pared cuando el robot se está alejando de la misma, el tercer if hace que el robot gire sobre si mismo hasta encontrar una pared a su derecha y en el resto de los casos se avanza en línea recta.

Comportamiento RodearAntiHorarios

Este comportamiento es análogo al anterior utilizando los sensores simétricos e intercambiando las velocidades de los motores.

Comportamiento Empujar

Este comportamiento se encarga de empujar el obstáculo hacia adelante utilizando únicamente los sensores infrarrojos del robot. Como se muestra en el Algoritmo 5 el robot realiza pequeños ajustes de velocidad tratando de colocarse en posición de empuje. El Algoritmo 5 define el comportamiento utilizado para empujar obstáculos hacia el home.

```
darAccion(in double sensoresIR[8], out double actuadores[2])
 const double VEL_AVANCE, AJUSTE, UMBRAL_DERECHO
begin
 if (sensoresIR[2]+sensoresIR[3])>UMBRAL_DERECHO)
   actuadores[0] = VEL_AVANCE
   actuadores[1] = VEL_AVANCE
 else if (sensoresIR[2]<sensoresIR[3])
   actuadores[0] = VEL_AVANCE
   actuadores[1] = VEL_AVANCE+AJUSTE
 else
   actuadores[0] = VEL_AVANCE+AJUSTE
   actuadores[1] = VEL_AVANCE
 end
end
```

Algoritmo 5. Comportamiento para empujar objetos.

7.2.2. Resultados

El comportamiento empujar al home se probó utilizando el simulador YAKS obteniendo buenos resultados. El comportamiento demostró ser robusto y comportarse de manera adecuada.

En la Figura 25 se muestra la configuración inicial del entorno utilizado para probar el comportamiento empujar al home. En la figura se puede apreciar en el vértice inferior derecho una zona marcada con un círculo que indica el home, sobre el vértice superior izquierdo 10 objetos a recolectar y en el vértice superior derecho el robot.

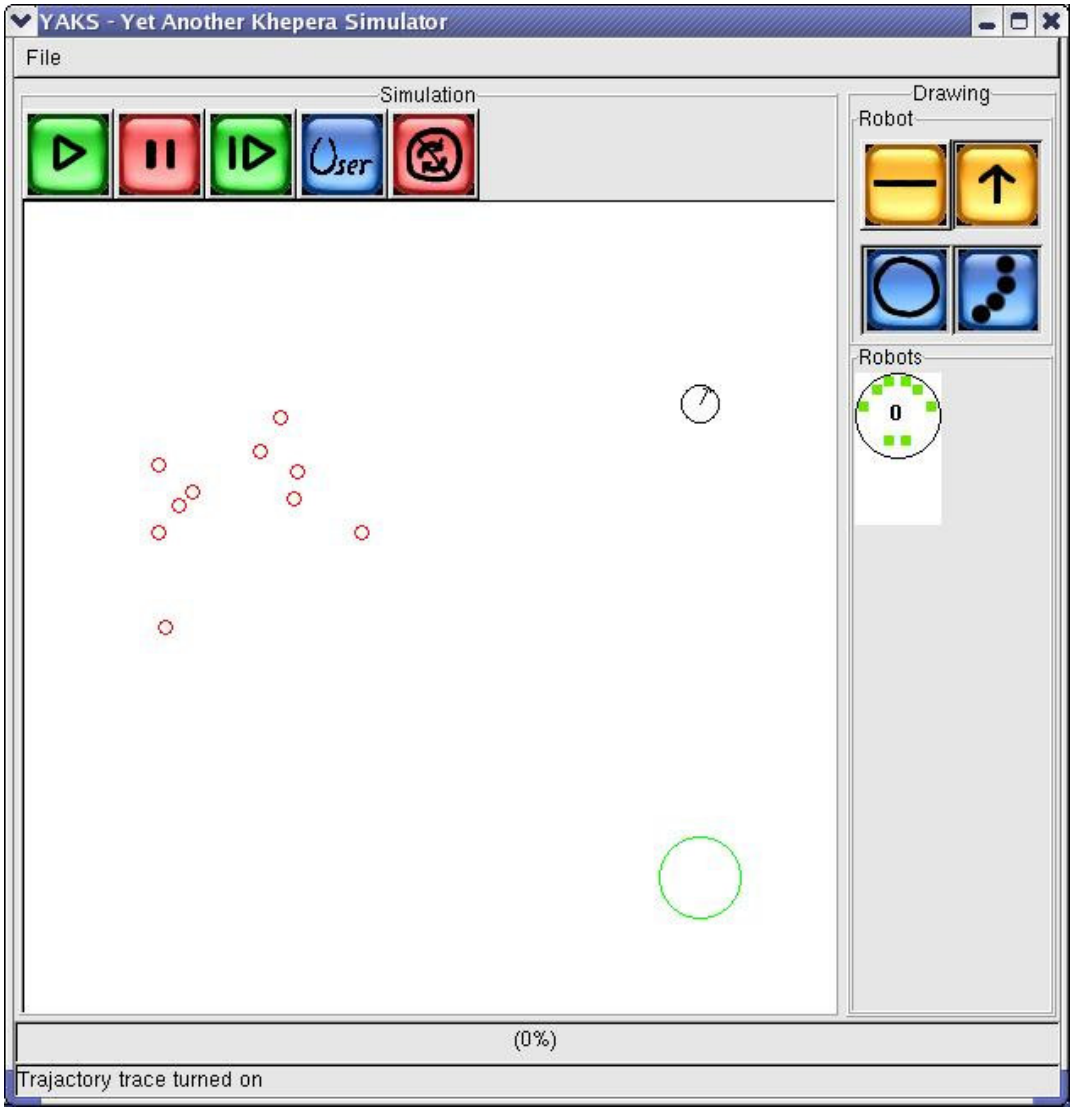


Figura 25. Estado inicial del entorno para un único robot.

En la Figura 26 se aprecia la trayectoria realizada por el robot desde el punto inicial, yendo a buscar el primer objeto a transportar y empujándolo hasta el home.

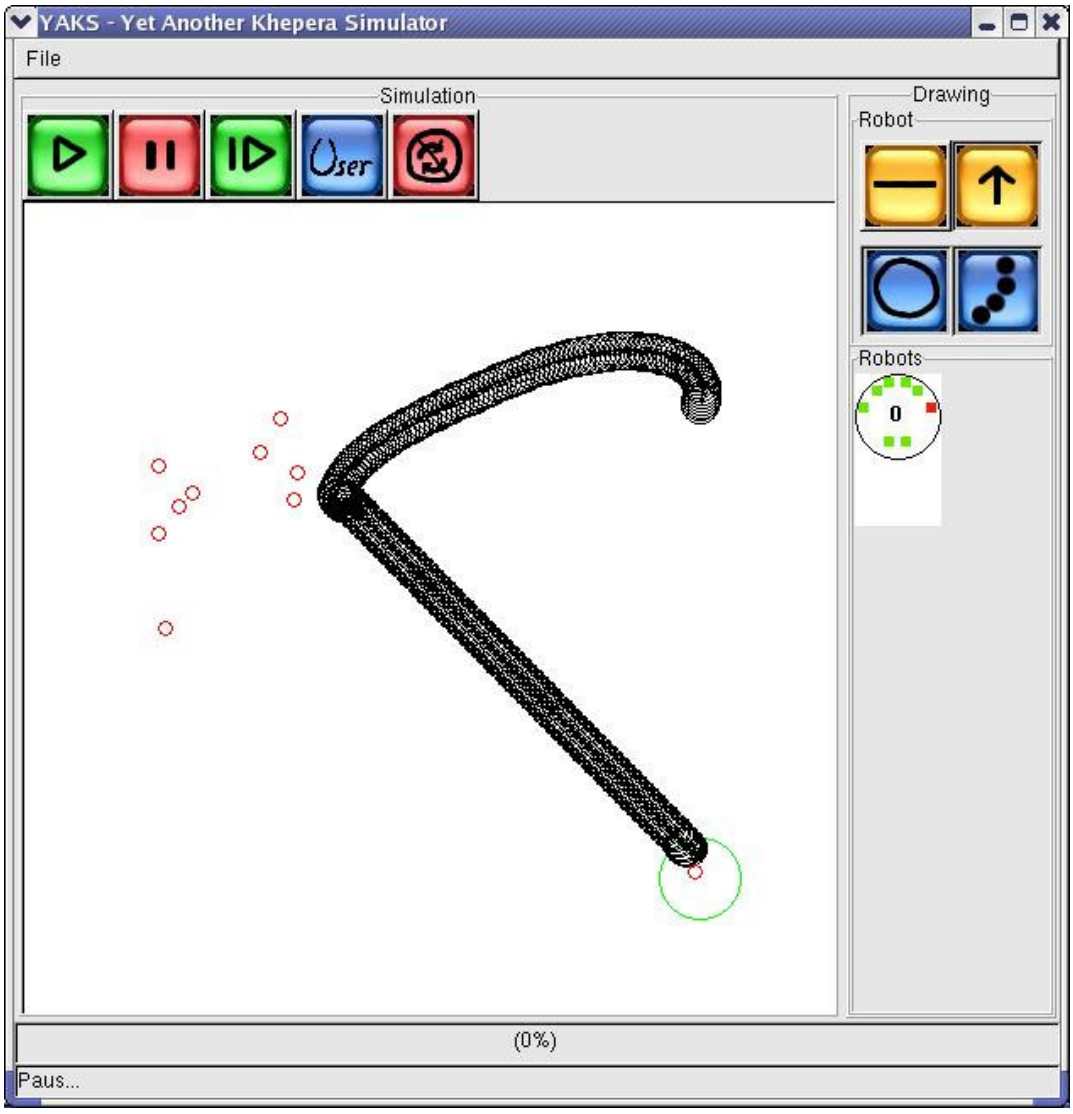


Figura 26. Trayectoria del robot empujando un objeto hasta el home.

Por último, en la Figura 27 se aprecia el fin de la simulación, todos los objetos fueron trasladados al home por el robot y este se queda explorando el mundo en busca de alimento.

Se puede apreciar en la misma imagen que hay un objeto fuera de la zona de home. Esta situación se debe a que el robot no puede percibir la posición de los objetos, esto es, el robot no puede determinar si el objeto está o no dentro del home. En la implementación se utiliza como aproximación para dejar de empujar el objeto transportado que el robot haya alcanzado el home, lo cual puede llevar a dejar objetos fuera de la zona de home.

En esta corrida el robot logra trasladar todos los objetos al home, lo cual no siempre ocurre pues la zona que explora el robot en busca de alimento está acotada, en caso de que un objeto este fuera de la zona de alimento no será trasladado por el robot.

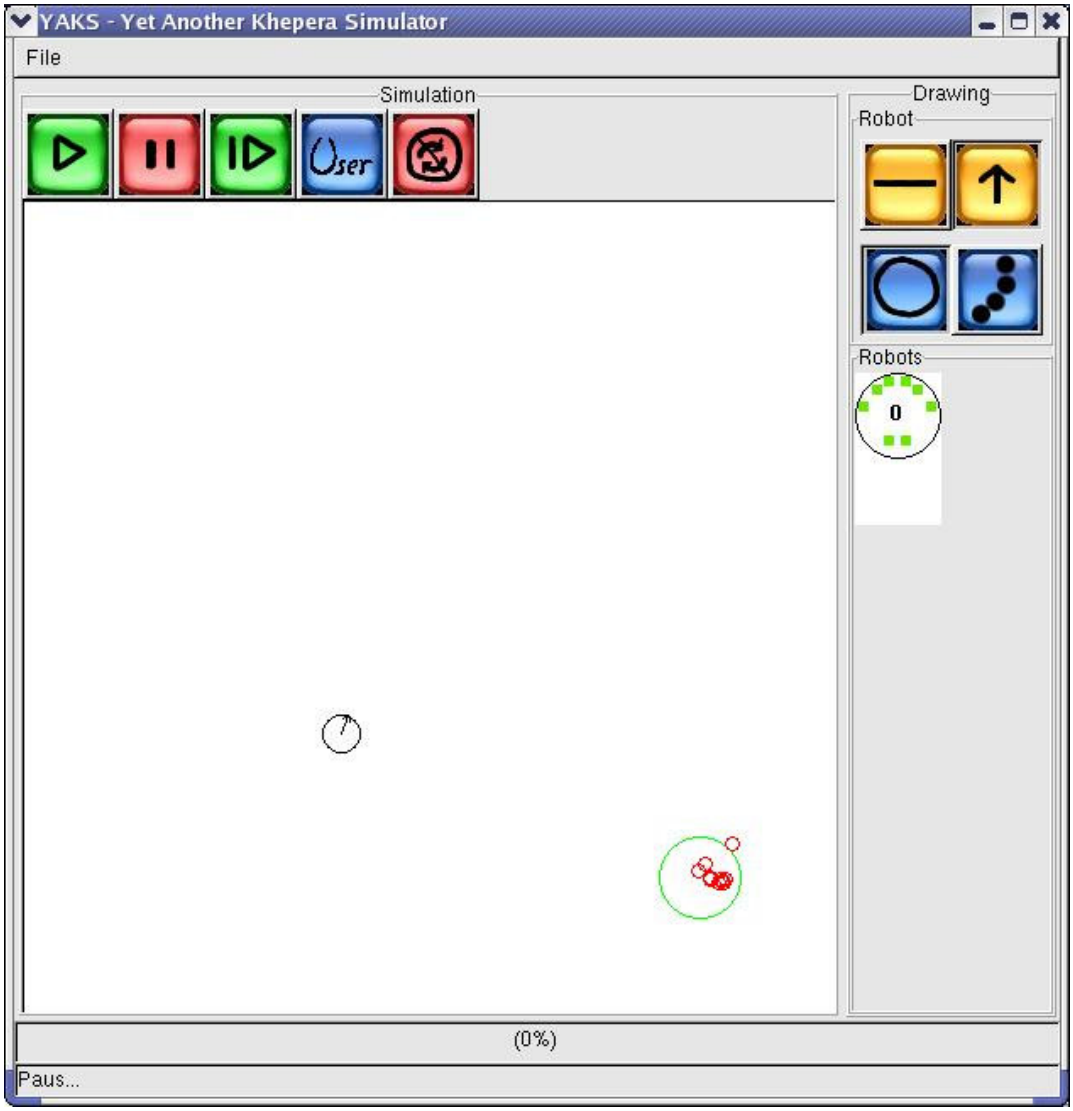


Figura 27. Estado final del entorno para un único robot.

7.3. Reportarse

La tarea de reportarse consiste en ir periódicamente hasta cierta posición en el mundo y emitir un reporte de la actividad realizada hasta el momento. Al igual que antes este comportamiento se modeló como una máquina de estados. En la Figura 28 se muestra la máquina de estados utilizada para implementar el comportamiento para reportar la actividad. En la misma se aprecian los comportamientos simples IrAZonaReporte y ReportarActividad, y los predicados tiempoDeReportar y finReporte.

El comportamiento IrAZonaReporte se implementó utilizando al igual que en el comportamiento IrPorComida, el algoritmo Bounded velocity Lyapunov (BV). El comportamiento ReportarActividad en lugar de emitir un reporte utilizando la comunicación inalámbrica para comunicarse con un usuario humano alejado de la zona donde se ejecuta la misión realiza una espera sin ninguna actividad.

Figura 28. Máquina de estados para el comportamiento Reportarse.

Los predicados se manejan generando eventos cada cierto tiempo, para el predicado tiempoDeReportar se genera un evento cada dos minuto y para predicado finReporte se genera un evento luego de diez segundos de activado el comportamiento ReportarActividad.

8. Conclusiones y Trabajos Futuros

8.1. Introducción

En este capítulo se presentan las conclusiones que se desprenden de este trabajo y las principales líneas de investigación para explorar a partir de lo realizado aquí.

8.2. Conclusiones

En este trabajo se estudió en profundidad la arquitectura ALLIANCE, detectando ambigüedades entre su definición formal y su definición en lenguaje natural. Se desarrolló un paquete de clases en el leguaje de programación Java simple, claro y extensible, que refleja con claridad entre sus clases los elementos definidos en ALLIANCE (comportamientos de alto nivel, supresores de actividad, comportamientos motivacionales y comunicación entre robots). Esto permitió verificar y reafirmar las bondades de ALLIANCE.

Se propuso un núcleo para ALLIANCE, el cual se ha denominado N-ALLIANCE. El mismo simplifica notoriamente al mecanismo de selección de tareas propuesto en ALLIANCE. Se eliminaron varios elementos de ALLIANCE que condicionaban innecesariamente las estrategias de selección que podían implementarse sobre ella. Esto, a su vez, hace al núcleo propuesto más simple y más potente. Sobre este núcleo se implementaron las arquitecturas ALLIANCE, L-ALLIANCE y las estrategias TC/N-ALLIANCE y CR/N-ALLIANCE. La implementación propuesta para L-ALLIANCE elimina varios parámetros que debía ajustar el diseñador y elimina las dos etapas propuestas por L-ALLIANCE. La etapa de aprendizaje activo requiere que todos los miembros del equipo trabajen juntos en una etapa previa a la misión, lo cual dificulta la inserción de nuevos robots al equipo durante la ejecución de una misión, reduciendo las bondades de ALLIANCE. Es por esto que N-ALLIANCE propone un mecanismo de intercambio de conocimiento entre los robots la primera vez que trabajan juntos, lo cual es más flexible permitiendo la inserción de robots dinámicamente durante la misión.

Se propuso dos nuevas estrategias de selección con el objetivo de mejorar el rendimiento del equipo. La primera denominada Task Coverage agrega consideraciones que toman en cuenta las tareas que sólo un robot o unos pocos robots saben realizar, de modo que estas tareas sean elegidas en primera instancia. Esta estrategia asigna a los robots las tareas para las cuales sólo él y unos pocos saben realizar. De esta forma el resto del equipo trabaja en paralelo en otras tareas. Si no se tomara en cuenta el task coverage podría ocurrir que un robot tome una tarea para ejecutar (que muchos robots saben realizar) y deje de ejecutar una tarea que sólo él puede realizar eliminado la posibilidad de que el resto del equipo trabaje en paralelo. Esta propuesta si bien parecía atractiva en principio, con las simulaciones se determinó que, en general, no conviene utilizarla pues se comporta peor que L-ALLIANCE resolviendo eficientemente unos pocos casos.

La segunda estrategia propuesta, denominada Cantidad de Robots, ataca un problema dejado de lado en L-ALLIANCE. Esta nueva estrategia incita a los robots a trabajar juntos en una misma tarea, tratando de distribuir uniformemente a los robots entre las distintas tareas que componen la misión. La misma mejora el rendimiento del equipo pero cuanto mayor sea la cantidad de robots respecto de la cantidad de tareas mayor es la interferencia entre robots.

Adicionalmente, se implementó un paquete genérico en el lenguaje de programación Java que permite aprender comportamientos utilizando el algoritmo Q-Learning. Este paquete requiere únicamente que el diseñador defina la función de refuerzo y un lugar

donde almacenar la función Q. Se realizaron varios experimentos utilizando tablas y redes neuronales para almacenar la función Q. Se comprobó que la técnica de aprendizaje por refuerzo es muy interesante desde el punto de vista de sus fundamentos biológicos, matemáticos y su uso práctico.

Se utilizó el simulador de robots YAKS sobre el cual se implementó una misión de recolección de residuos. Se corrieron varias misiones utilizando las diferentes estrategias y se verificó un mejor rendimiento al utilizar las estrategias propuestas. En particular, se comprobó que la estrategia CR/N-ALLIANCE se comporta mejor que L-ALLIANCE y TC/N-ALLIANCE.

8.3. Trabajos Futuros

Es de interés desarrollar una interfaz que permita interactuar de forma más amigable con la implementación de ALLIANCE. En particular sería interesante disponer de una interfaz que:

- permita consultar los parámetros en tiempo real de la arquitectura,
- definir y construir la arquitectura de comportamientos de alto nivel,
- enviar dinámicamente comportamientos a los robots.

Todos los experimentos fueron realizados sobre el simulador de robots YAKS y sería deseable verificar el caso de estudio utilizando robots reales.

Resulta un gran desafío estudiar la forma de utilizar N-ALLIANCE en entornos cooperativos con adversarios, en particular fútbol de robots.

Sería interesante disponer de un mecanismo que aprenda a ajustar los parámetros de N-ALLIANCE de forma de mejorar el rendimiento del equipo en lugar de utilizar estrategias fijas (L-ALLIANCE, TC-ALLIANCE o CR-ALLIANCE) que adaptan dichos parámetros dinámicamente, o sea, un mecanismo que adapte estrategias.

Glosario

Adaptabilidad: capacidad de reaccionar a cambios internos o externos.

AEP: en [Parker1995] se propone un problema similar al MRTA denominado Problema de la Eficiencia de ALLIANCE (ALLIANCE Efficiency Problem) (AEP).

AFNOR: Asociación Francesa de Normalización (Association Française de Normalisation) [AFNOR2004].

Agente: entidad que percibe el mundo y actúa en él.

ALLIANCE: arquitectura de software cooperativa utilizada para asignar tareas a robots de forma robusta, definida en [Parker1998].

Autómata: máquina que imita la figura y movimientos de un ser animado.

Ciencias de la vida: cualquiera de las ramas de la biología que trata con la estructura o comportamiento de los organismos vivos.

Comportamiento: ley de control que realiza o mantiene cierto objetivo.

Comportamiento emergente: se manifiesta como un estado global o patrón en el tiempo que no está explícitamente programado y se obtiene como resultado de las interacciones locales entre los componentes del sistema.

Comportamiento reactivo: comportamiento que acopla fuertemente el sistema sensorial con el sistema motor, no utiliza representaciones abstractas ni un historial para actuar.

Cooperación: Acción y efecto de cooperar.

Cooperar: Obrar juntamente con otro u otros para un mismo fin.

CR/N-ALLIANCE: estrategia desarrollada para controlar los parámetros de la arquitectura N-ALLIANCE, con el objetivo de asignar de manera uniforme los robots a las tareas, esta estrategia se define en el punto 5.3.2 de este trabajo.

Etología: rama de la zoología que estudia el comportamiento de los animales en su hábitat natural.

IFR: Federación Internacional de Robótica (International Federation of Robotics) [IFR2004].

Interferencia: es cualquier influencia que bloquea o se opone al comportamiento ejecutado por el agente.

ISO: Organización Internacional de Estándares (International Standard Organisation) [ISO2004]

K-Team: es el fabricante del robot Khepera.

Khepera: pequeño robot móvil desarrollado por la empresa K-Team [K-Team1999a].

L-ALLIANCE: arquitectura de software cooperativa utilizada para asignar tareas a robots de forma robusta y eficiente, definida en [Parker1995].

L/N-ALLIANCE: implementación de la estrategia L-ALLIANCE sobre N-ALLIANCE.

MRTA: nombre utilizado para el problema de asignación de tareas a múltiples robots (Multi-Robot Task Allocation) (MRTA).

N-ALLIANCE: arquitectura de control basada en las ideas principales de ALLIANCE, más simple y flexible. Esta arquitectura se define en el punto 4.2 de este trabajo.

PWW: condición definida en [Parker1995] que en caso de cumplirse asegura que cuando un robot trabaja en una tarea durante ε (ε>0) unidades de tiempo, el tiempo requerido para la finalización de la misión se decrementa.

RIA: Asociación de Industrias Robóticas (Robotic Industries Association) [RIA2004]

Robot: criatura mecánica capaz de funcionar de manera autónoma [Murphy2000].

TC/N-ALLIANCE: estrategia desarrollada para controlar los parámetros de la arquitectura N-ALLIANCE. Se basa en la estrategia L-ALLIANCE con la inclusión del task coverage al momento de realizar la asignación. Esta estrategia se define en el punto 5.3.1 de este trabajo.

YAKS: simulador del robot Khepera (Yet Another Khepera Simulator) [Zaxmy2003].

Zoología: rama de la biología que estudia los animales.

Bibliografía

[AFNOR2004] AFNOR, Estándar NF E 61-100, http://www.afnor.fr, visitada en julio de 2004.

[Arkin1998] Arkin R. C., Behavior-Based Robotics, Mit Press, Mayo 1998.

[Asimov1950] Asimov I., I Robot, Gnome Press, 1950.

[Balch2000] Balch T., "Hierarchic social entropy: an information theoretic measure of robot team diversity", Autonomous Robots, 8:3, 2000.

[Barrientos1997] Barrientos A., L. F. Peñin, C. Balaguer y R. Aracil, Fundamentos de Robótica, McGraw-Hill, 1997.

[Bergfeldt2000] Bergfeldt N., "Cooperative Robotics: A Survey", M.Sc. Dissertation, Department of Computer Science,University of Skövde, SWEDEN, 2000.

[Braitenberg1985] Braitenberg V., Vehicles, Mit Press, 1985.

[Brooks1986] Brooks, R. A., "A Robust Layered Control System for a Mobile Robot", IEEE Journal of Robotics and Automation, RA-2, pp.14-23, 1986.

[Brooks1989] Brooks, R. A. y A. M. Flynn, "Fast, Cheap and Out of Control: A Robot Invasion of the Solar System", Journal of the British Interplanetary Society, pp. 478--485, 1989.

[Cao1995] Cao Y. U., Fukunaga A. S. y A. B. Kahng, "Cooperative Mobile Robotics: Antecedents and Directions", In Proc. of IEEE Int. Conf. on IROS, 1995.

[Cavallaro1994] Cavallaro, J. R. y I. D. Walker, "A Survey of NASA and Military Standards on Fault Tolerance and Reliability Applied to Robotics", In Proc. AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space, pp. 282-286, Houston, TX, 1994.

[CRL2004] Central Research Laboratorios, "History of Telemanipulator Development", http://www.centres.com/nuclear/manip/maniphis.htm, visitada en marzo de 2004.

[Dayhoff1990] Dayhoff J., Neural Network Architectures: An Introduction, Van Nostrand Reinhold, 1990.

[Gerkey2003] Gerkey B. P., "On Multi-Robot Task Allocation", PhD Dissertation. University of Southern California Computer Science Department, 2003.

[Haykin1994] Haykin S., Neural Network: A Comprehensive Foundation, Macmillan, 1994.

[Herbert1984] Herbert F., Dune, Ultramar, 1984.

[Hertz1991] Hertz J.A., A. Krogh y R.G. Palmer, Introduction to the Theory of Neural Computation, Addison Wesley, 1991.

[IFR2004] IFR, Página principal de la Federación Internacional de Robótica, http://www.ifr.org/, visitada en junio de 2004.

[ISO1994] ISO, Manipulating Industrial Robots, Estándar ISO 8373:1994, 1994.

[ISO2004] ISO, Página principal de la Organización Internacional de Estándares, http://www.iso.org/, visitada en junio de 2004.

[Jung1999] Jung D. y A. Zelinsky, "An Architecture for Distributed Cooperative-Planning in a Behaviour-based Multi-robot System", Journal of Robotics and Autonomous Systems (RA&S) 26, pp. 149-174, 1999.

[K-Team1999a] K-Team, "Khepera User Manual", Version 5.02, 1999.

[K-Team1999b] K-Team, "Khepera Radio Base User Manual", Version 1.0, 1999.

[K-Team1999c] K-Team, "Khepera Radio Turret User Manual", Version 1.0, 1999.

[Kaelbling1996] Kaelbling L., M. Littman, and A. Moore. "Reinforcement learning: A survey". Journal of Artificial Intelligence Research, 4: pp. 237-285, 1996.

[Kube1992] Kube C. R. y H. Zhang. "Collective robotic intelligence". In Proceedings of the Second International Workshop on Simulation of Adaptive Behavior, pp. 460-468, 1992.

[Kube1993] Kube C. R. y H. Zhang. "Collective robotics: From social insects to robots". Adaptive Behavior, 2(2): pp. 189-219, 1993.

[Mataric1992] Mataric M. J., "Minimizing Complexity in Controlling a Mobile Robot Population", in Proceedings, IEEE International Conference on Robotics and Automation, pp. 830-835, Nice, France, 1992.

[Mataric1994] Mataric M. J., "Learning to Behave Socially", MIT Artificial Intelligence Laboratory, 1994.

[Mataric1995] Mataric M. J., "Designing and Understanding Adaptive Group behavior", Adaptive Behavior, 4:1, 1995.

[Mataric1997] Mataric M. J., "Learning social behaviors", Robotics and Autonomous Systems, 20, pp. 191-204, 1997.

[Mataric1998] Mataric M. J., "Coordination and Learning in Multi-Robot Systems", IEEE Intelligent Systems, pp. 6-8, 1998.

[Mitchel2004] Mitchel O., “Khepera Simulator Version 2 User Manual”, http://diwww.epfl.ch/lami/team/michel/khep-sim/, visitada en julio de 2004.

[Moody1989] Moody J. y C. J. Darken, "Fast Learning in Networks of Locally-Tuned Processing Units", Neural Computation, 1, pp. 281-294, 1989.

[Murphy2000] Murphy R. R., An Introduction to AI Robotics (Intelligent Robotics and Autonomous Agents), MIT Press, 2000.

[Parker1997] Parker L.E., "L-ALLIANCE: Task-oriented multi-robot learning in behavior-based systems", Journal of Advanced Robotics, 1997.

[Parker1998] Parker L. E., "ALLIANCE: an architecture for fault tolerance multi-robot cooperation", IEEE Transactions on Robotics and Automation, 1998.

[Parker1999] Parker L. E., "Cooperative robotics for multi-target Observation", Intelligent Automation and Software Computing. 1999.

[Parker2000] Parker L. E., "Current State of the Art in Distributed Autonomous Mobile Robotics", in Distributed Autonomous Robotic Systems 4, L. E. Parker, G. Bekey, and J. Barhen eds., Springer-Verlag, pp. 3-12, 2000.

[RIA2004] Robot Industries Association, http://www.roboticsonline.com/, visitada en Julio de 2004.

[Santos1999] Santos J. M. Y C. F. Touzet, "Dynamic Update of the Reinforcement Function During Learning", Connection Science, 11, pp. 3-4, 267-289, 1999.

[Sutton1998] Sutton R. S. y A. G. Barto, Reinforcement Learning: An Introduction, Mit Press, 1998.

[Touzet1997] Touzet C. F., "Neural reinforcement learning for behavior synthesis", Robotics and Autonomous Systems,1997.

[Touzet1998] Touzet C. F., "Bias Incorporation in Robot Learning" Autonomous Robots, 2, pp. 1-18, 1998.

[Webers2002] Webers C. y U. R. Zimmer, “Motion Control of Mobile Robots: from static targets to fast drives in moving crowds”, Autonomous Robots, Vol. 12, No. 3, 2002.

[Werger1999] Werger B. B., "Cooperation without deliberation: A minimal behavior-based approach to multi-robot teams". Artificial Intelligence, 110:293--320, 1999.

[Zaxmy2003] Zaxmy J., YAKS: Yet Another Khepera Simulator, http://r2d2.ida.his.se/, setiembre 2003.

Apéndice A. El Simulador YAKS

A.1. Introducción

En este apéndice se presentan algunos de los simuladores evaluados para ser utilizados. YAKS fue el simulador utilizado para realizar los experimentos. Sobre el mismo se realizaron modificaciones con el objetivo de permitir que YAKS interactúe con Java.

A.2. Evaluación de Simuladores

Al no disponer de un robot real para realizar los experimentos es necesario trabajar con un simulador de robot. Se probaron varios simuladores y se optó por utilizar YAKS [Zaxmy2003]. Las principales ventajas que presenta este simulador son: ser gratuito, disponer del código fuente, que corre en diversas plataformas, y ser un simulador del robot Khepera. Que sea un simulador del robot Khepera es una característica importante pues como se indicó en el Capítulo 2 este robot es ampliamente utilizado en las investigaciones y se trabajó con este robot en aprendizaje de comportamientos simples.

Al trabajar con este simulador se observa que no controla la interacción entre un robot y los distintos elementos del entorno, esto es, el simulador permite que un robot atraviese paredes, obstáculos y otros robots.

Permite trabajar con varios robots simultáneamente cada uno con su propio comportamiento, lo cual es un requerimiento necesario para este trabajo y algunos simuladores no lo cumplen [Mitchel2004].

Es importante hacer notar que existe un simulador 3D, denominado Webots, desarrollado por la misma empresa que construyó el robot pero su costo es prohibitivo, más aún si se tiene en cuenta que YAKS es gratuito.

A.3. Simulador YAKS

El entorno se define mediante un archivo de texto, el mismo puede incluir paredes, obstáculos redondos, zonas y focos de luz. Para cada uno de estos elementos es necesario definir su posición y su dimensión.

El simulador permite definir la potencia de cada motor, indicando un valor entre cero (máxima velocidad de retroceso) y uno (máxima velocidad de avance), y conocer en todo momento la lectura de los sensores. En la Figura 29 se muestra la interfaz proporcionada por YAKS y se muestran varios de los objetos al utilizar un mundo tipo laberinto que trae por defecto definido el simulador. En dicha figura se puede apreciar el robot como un círculo con una pequeña flecha interior que marca su frente, las paredes y varios obstáculos como círculos más pequeños. La interfaz gráfica del simulador permite, entre otras cosas, pausar la simulación, avanzar la simulación, eliminar la actualización visual (esto acelera la simulación), modificar el aspecto del robot, conocer la lectura de los sensores y dejar una huella de la trayectoria recorrida por los robots.

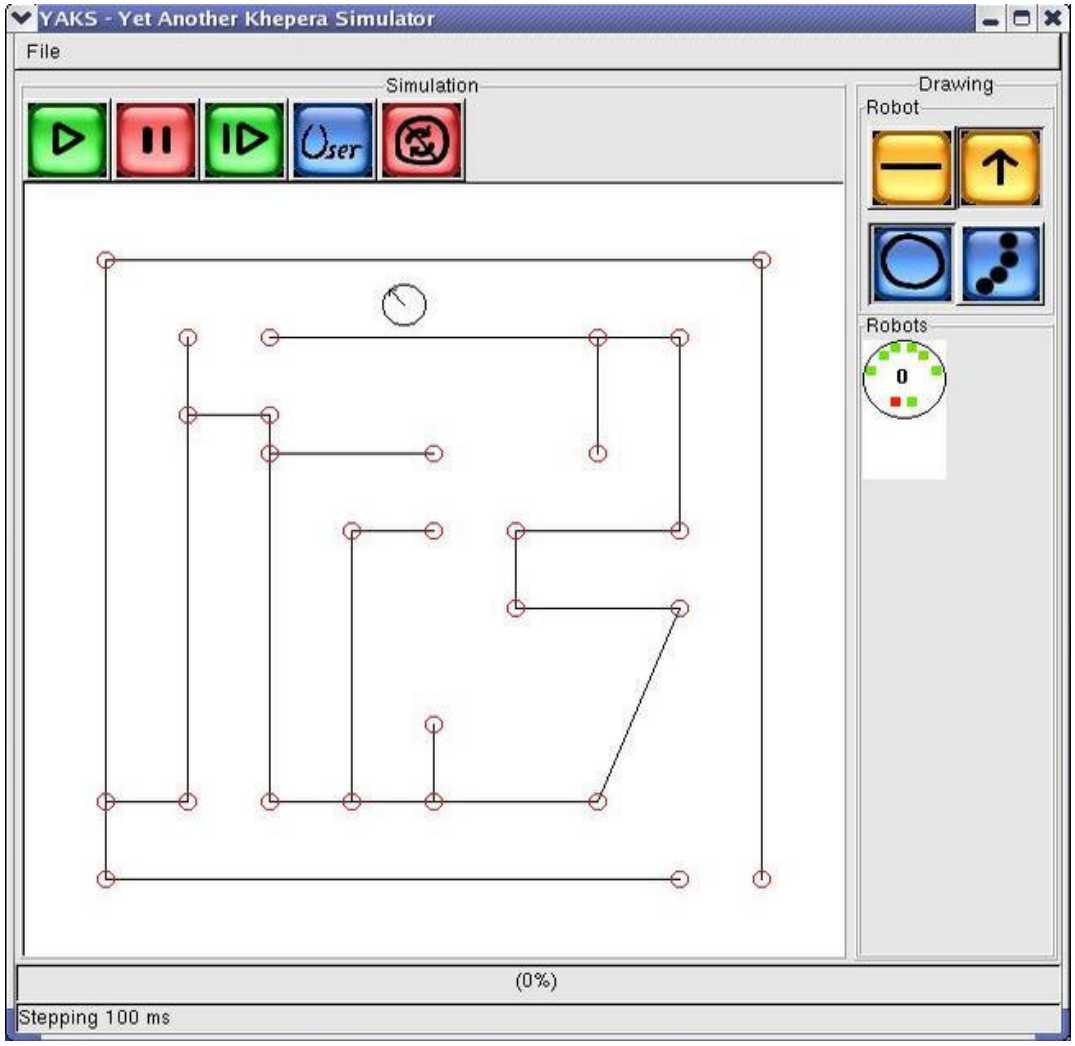


Figura 29. Interfaz del simulador YAKS.

A.4. Modificaciones realizadas

Como se indicó antes el simulador utiliza un modelo muy reducido de la realidad permitiendo incluso atravesar paredes, objetos y robots.

Para trabajar adecuadamente con el simulador y poder resolver problemas interesantes, tratando de hacerlo más real, se incorporó al simulador la lógica de control necesaria para empujar objetos, no atravesar robots y no atravesar paredes.

A.4.1. Servidor YAKS

Se desarrollo un servidor de simulación en el lenguaje de programación C. El mismo permite que varios clientes (robots remotos) se conecten a él permitiendo simular el comportamiento de estos robots remotos en un entorno 2D.

Para dicha implementación se tomo el código de YAKS, utilizándolo como parte central del servidor TCP/IP y se implementó un API Java que interactúa con dicho servidor, permitiendo realizar las siguientes operaciones:

- Conectarse a un servidor.
- Ajustar los motores de un determinado robot.
- Consultar los valores de los sensores.
- Modificar la posición y dirección de un determinado robot en el entorno.
- Obtener la posición y dirección de un determinado robot en el entorno.
- Modificar la posición de un determinado obstáculo en el entorno.
- Obtener la posición de un determinado obstáculo en el entorno.
- Ajustar la amplitud y la altura de la pinza.
- Consultar la cantidad de obstáculos y las dimensiones del mundo

Todas estas operaciones se implementaron como RPC, donde los clientes empaquetan el código de operación y los parámetros requeridos, y le envían al servidor de simulación la solicitud. Luego si corresponde el servidor envía al cliente el resultado de la consulta.

El servidor de simulación se implementó en el lenguaje de programación C utilizando sockets para la comunicación con los clientes. El servidor permite que diversos clientes se conecten a él mediante un protocolo muy simple de intercambio de mensajes sobre una red TCP/IP, esto permite simular el comportamiento de un robot desde una máquina remota y utilizando diversos lenguajes de programación. Al permitir que los clientes corran en máquinas remotas alivia notoriamente la carga de la máquina en la que corre el servidor si se compara con otras alternativas que permiten integrar diversos lenguajes.

El servidor de simulación YAKS se desarrolló en C++ utilizando el entorno de desarrollo Kdevelop 2.1 sobre Linux RedHat 9.0. Para este desarrollo se implementaron aproximadamente 2110 líneas de código C++.

Apéndice B. Diseño del Software

B.1. Introducción

En este apéndice se presentan los paquetes, las clases y las interfaces más importantes desarrollados en este trabajo. En el mismo se implementaron aproximadamente 8600 línea de código Java utilizando como plataforma de desarrollo Eclipse 2.1 en Linux Red Hat 9.0. En la última parte de este apéndice se puede encontrar el código completo utilizado necesario para que un robot ejecute el caso de estudio controlado por la estrategia CR/N-ALLIANCE.

En la Figura 30 se muestran los paquetes desarrollados en Java.

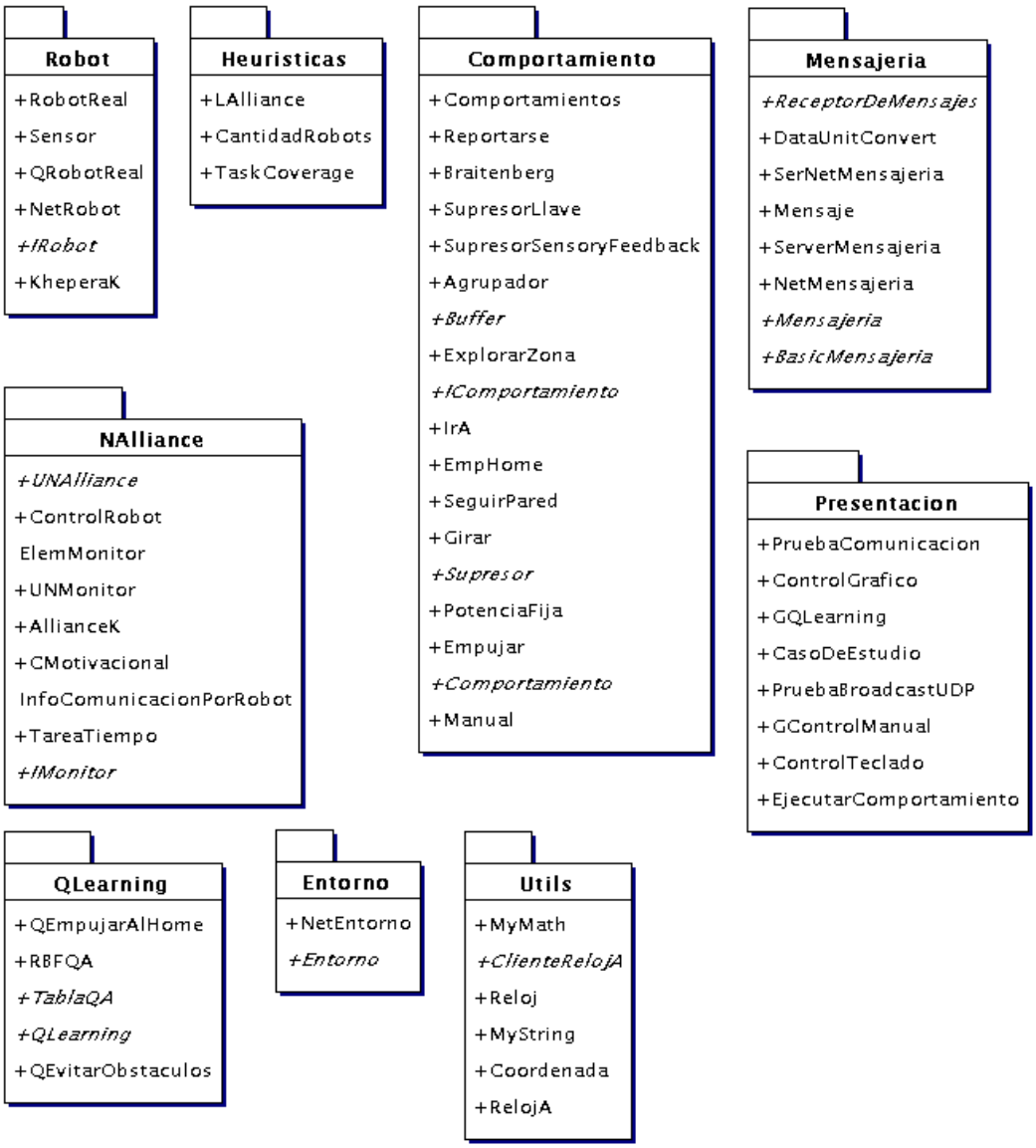


Figura 30. Paquetes desarrollados en Java.

B.2. Servicio de mensajes

El servicio de mensajería brinda las funcionalidades necesarias para que un equipo de robots pueda intercambiar información de actividad durante la realización de una misión. En la Figura 31 se muestra el diagrama de clases del paquete Mensajeria.

B.2.1. Mensaje

La definición de ALLIANCE establece que los robots deben intercambiar información cada ρ unidades de tiempo. Por otro lado, es necesario definir qué tipo de información requieren intercambiar los robots. La clase Mensaje define la unidad de intercambio de información entre los robots que participan de la misión.

La clase Mensaje incluye los siguientes atributos:

- idRobotOrigen: este atributo permite conocer el identificador del robot que envió el mensaje.
- idTarea: este atributo permite conocer el identificador de la tarea respecto de la cual se desea informar.
- MSG_TIPO_FIN: constante para el atributo tipo de mensaje asociada a un mensaje que indica la finalización de la tarea.
- MSG_TIPO_INFO_COMP: constante para el atributo tipo de mensaje asociada a un mensaje usado para informar cambios en los comportamientos de un robot.
- MSG_TIPO_OCIOSO: constante para el atributo tipo de mensaje asociada a un mensaje que indica que el robot está ocioso.
- MSG_TIPO_TRABAJANDO: constante para el atributo tipo de mensaje asociada a un mensaje que indica que el robot está trabajando.
- stamp: este atributo es utilizado para almacenar información de tiempo referente a la tarea que se informa.
- tipo: es el tipo de mensaje generalmente asociado al estado de ejecución del robot.

B.2.2. El paquete Mensajeria

Mensajeria

Esta interfaz presenta las bases para la construcción de los distintos mecanismos de mensajería utilizados en las distintas etapas del desarrollo de ALLIANCE.

Los métodos definidos en esta interfaz son:

- enviarMensaje: este método es utilizado para enviar un mensaje a todos los robots del equipo.
- desSuscribirse: este método permite eliminar la suscripción de un elemento al servicio de mensajería.
- subscribirse: este método permite suscribir un elemento al servicio de mensajería.

Las clases que implementan esta interfaz deben garantizar que todo mensaje enviado utilizando el método *enviarMensaje* tiene como destino a todos los elementos subscriptos.

Desde el punto de vista del comportamiento tanto la recepción como el envío de mensajes debe ser asíncrona y no bloqueante.

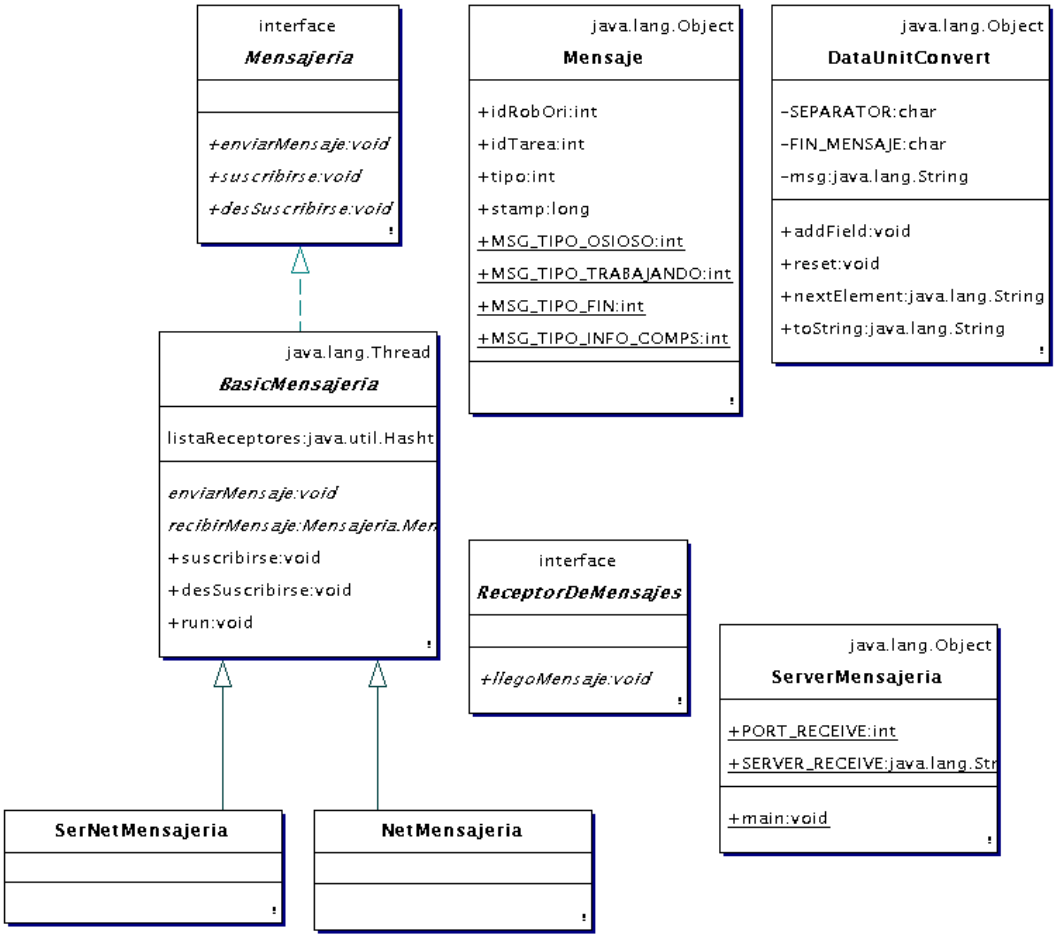


Figura 31. Diagrama de clases para el paquete Mensajería.

El servicio de mensajería se construye a partir de la interfaz Mensajería, la cual define los métodos esenciales que debe implementar un servicio de mensajería. Se implementaron tres tipos de servicio a partir de la interfaz Mensajeria, SoftMensajeria, NetMensajeria y SerNetMensajeria, que se explican a continuación

BasicMensajeria

Esta clase implementa todas las operaciones independientes del servicio de mensajería que se utiliza.

El atributo listaReceptores es una tabla de hash con todos los clientes del servicio de mensajería. Los clientes de este servicio sólo deben implementar la interfaz ReceptorDeMensajes.

Esta clase implementa las operaciones suscribirse y desSuscribirse que agregan y eliminan respectivamente de la lista de receptores.

Para poder recibir mensajes sin necesidad de bloquear al proceso que utiliza al servicio de mensajería la clase BasicMensajeria extiende la clase Thread. El método run del Thread se queda esperando mensajes y toda vez que recibe un mensaje informa a todos los clientes subscriptos.

Software de mensajería

Esta implementación fue la primera en desarrollarse, principalmente pensando en eliminar las dificultades de utilizar un medio de comunicaciones real (errores en la comunicación y latencia) y el paralelismo, permitiendo probar la implementación de ALLIANCE independiente de estos asuntos.

Esta implementación facilitó la construcción del primer prototipo de ALLIANCE donde todos los robots del equipo corrían en un mismo proceso de manera secuencial.

Servidor de mensajería (ServerMensajeria)

La siguiente implementación utiliza un proceso servidor de mensajería al cual los robots del equipo se registran para recibir mensajes. Luego cuando un robot desea enviar información de su actividad envía un mensaje al servidor y éste se encarga de realizar su distribución a todos los robots registrados. Por último, los robots reciben la información del servidor y actualizan la información almacenada de comunicación.

Este modelo presenta al servidor como un punto único de falla por lo que no puede utilizarse en misiones reales, pero presenta un entorno de prueba más real utilizando una red de datos real para transmitir los mensajes. Las pruebas de ALLIANCE realizadas utilizando este servicio permiten correr cada instancia de ALLIANCE en un proceso diferente corriendo incluso en diferentes máquinas.

Los robots utilizan a la clase SerNetMensajeria para hacer uso del servicio de mensajería e interactuar con el servidor de mensajes.

NetMensajeria

Esta última implementación permite que los robots intercambien mensajes utilizando los mecanismos de broadcast implementados en los servicios de red (TCP/IP), lo cual elimina la necesidad de utilizar un servidor de mensajería. El servicio de mensajes, en este caso NetMensajería, permite que un mensaje sea enviado directamente a todos procesos que estén escuchando en la red.

Esta implementación se adecua a la realidad de un equipo de robots, cada uno con su sistema de transmisión inalámbrico corriendo ALLIANCE y utilizando los métodos de la clase NetMensajería para comunicarse con sus compañeros.

B.3. Comportamiento

La clase Comportamiento implementa los métodos que todo comportamiento debe incluir para interactuar con la arquitectura ALLIANCE. En la Figura 32 se muestra la clase comportamiento en la cual se pueden apreciar los principales métodos.

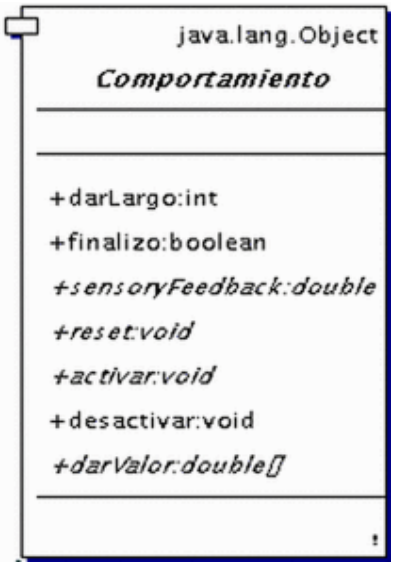


Figura 32. La clase Comportamiento.

Los métodos definidos en la clase comportamiento son:

- activar: es invocado toda vez que se desea informar al comportamiento que comenzó a utilizarse.
- darLargo: devuelve la cantidad de sensores que utiliza el comportamiento como entrada.
- darValor: es el método más importante de la clase, se encarga de implementar el comportamiento retornando la acción a ejecutar.
- desactivar: es invocado toda vez que se desea informar al comportamiento que dejó de utilizarse.
- finalizo: devuelve verdadero sí el comportamiento finalizó la tarea que debe implementar. Por ejemplo el comportamiento recolectar debe retornar verdadero cuando llegó al home con el objeto.
- reset: es invocado toda vez que se desea informar al comportamiento que comenzó a ejecutarse una nueva misión.
- sensoryFeedback: implementa la función con igual nombre propuesta por Parker. Para las tareas de recolección esta función retorna uno desde el principio de la misión hasta que el robot detecta que no hay más objetos que recolectar, a partir de ese momento retorna cero.

Se decidió incorporar función sensoryFeedback al comportamiento por ser ésta la clase más adecuada para implementarlo.

B.4. El paquete NALLIANCE

Figura 33. Diagrama de clases para el paquete NALLIANCE.

B.4.1. Comportamiento motivacional (CMotivacional)

Los comportamientos motivacionales implementan las funciones acquiescence, commReceived, end, motivation, taskCoverage y workingTime definidos en la arquitectura N-ALLIANCE. El resto de los métodos se utilizan en la interacción con NALLIANCE.

Los métodos definidos en la clase comportamiento son:

- activate: es invocado por NALLIANCE para indicarle al comportamiento motivacional que ha sido elegido para ejecutar.
- actualizar: es invocado por NALLIANCE para indicarle que debe actualizar la información de motivación, esto es, calcular el nuevo valor para la motivación según la recurrencia definida.
- acquiescence: implementa la función *acquiescence* definida en la arquitectura N-ALLIANCE.
- commReceived: implementa la función *commReceived* definida en la arquitectura N-ALLIANCE.
- deactivate: es invocado por NALLIANCE para indicarle al comportamiento motivacional que ha sido desactivado.
- deleteRobot: es invocado por NALLIANCE para indicarle al comportamiento motivacional que un determinado robot ha abandonado el equipo.
- end: implementa la función *end* definida en la arquitectura N-ALLIANCE.
- mensajeRecibido: es invocado por NALLIANCE toda vez que llega un mensaje que debe ser procesado por el comportamiento motivacional.
- motivation: implementa la función *motivation* definida en la arquitectura N-ALLIANCE.
- newRobot: es invocado por NALLIANCE para indicarle al comportamiento motivacional que un nuevo robot se ha sumado al equipo.
- taskCoverage: implementa la función *taskCoverage* definida en la arquitectura N-ALLIANCE.
- workingTime: implementa la función *workingTime* definida en la arquitectura N-ALLIANCE.

Esta clase tiene los métodos addRobot y deleteRobot que son invocados por N-ALLIANCE toda vez que determina que un robot se agregó al equipo o dejó de funcionar respectivamente. Además, cada comportamiento motivacional sólo almacena información de los robots que saben ejecutar la tarea asociada a él permite un manejo dinámico y eficiente de la memoria.

Esta clase tiene los siguientes atributos:

- comAltoNivel: este atributo contiene al comportamiento de alto nivel controlado por el comportamiento motivacional. Es usado por el comportamiento motivacional para conocer si corresponde ejecutar el comportamiento, sí finalizó de ejecutar la tarea, y para informarle que se activó o desactivó.
- delta: almacena el valor de *delta* definido en N-ALLIANCE que determina el grado de impaciencia.
- idTarea: almacena el identificador de la tarea asociada a dicho comportamiento.
- lambda: almacena el valor de *lambda* definido en N-ALLIANCE que determina el grado de asentimiento.
- motivacion: almacena la motivación del conjunto motivacional.

- supresorConjComp: permite controlar el supresor de actividad asociado al comportamiento motivacional.

B.4.2. Monitores de actividad (UNMonitor)

Los monitores de actividad fueron agregados a la arquitectura ALLIANCE para monitorear la comunicación y extraer de esta información el rendimiento. Se asocia un monitor a cada comportamiento motivacional. Su función principal es responder el tiempo que todo robot del equipo demora en ejecutar la tarea asociada que pueda realizarla. La clase UNMonitor implementa las funcionalidades asignadas a los monitores de actividad. En la Figura 33 se puede apreciar dicha clase, sus métodos y las relaciones que existen con otras clases del paquete NALLIANCE.

Los monitores de actividad calculan el tiempo de ejecución como un promedio de las últimas μ ejecuciones de la tarea más una desviación estándar. Para esto deben recibir, vía comunicaciones toda la información. Si el robot termina de ejecutar la tarea con éxito se almacena el tiempo de ejecución en caso contrario se penaliza multiplicando el tiempo de ejecución estimado por un factor mayor a uno. Es importante aclarar que no alcanza con que los monitores reciban sólo la información asociada a la tarea que se encarga de monitorear, sino deben conocer cuando un robot está ocioso o pasó a ejecutar otra tarea y ajustar el tiempo de ejecución estimado según corresponda.

Los métodos definidos en la clase UNMonitor son:

- deleteRobot: este método es invocado por NALLIANCE toda vez que determina que un robot abandonó el equipo.
- llegoMensaje: este método es invocado por NALLIANCE toda vez que debe comunicarle un mensaje necesario para calcular los tiempos de ejecución.
- taskTime: implementa la principal funcionalidad de los monitores de actividad, la función *taskTime* definida en N-ALLIANCE.
- taskCategory: implementa la función *taskCategory* definida en N-ALLIANCE.

Esta clase tiene los siguientes atributos:

- idTarea: almacena el identificador de la tarea que debe monitorear.
- mu: almacena el valor de μ definido en N-ALLIANCE.
- penalty: almacena el factor de penalización aplicado al tiempo de ejecución cuando un robot falla al ejecutar la tarea monitoreada.
- TASK_CATEGORY_1: almacena el valor devuelto por la función taskCategory cuando la tarea monitoreada pertenece a la categoría 1 definida en ALLIANCE.
- TASK_CATEGORY_2: almacena el valor devuelto por la función taskCategory cuando la tarea monitoreada pertenece a la categoría 2 definida en ALLIANCE.

B.4.3. Supresores de actividad

Los supresores de actividad son incluidos para componer los comportamientos utilizando la propuesta de Brooks [Brooks1986]. Se utilizan principalmente por los comportamientos motivacionales para activar o desactivar el comportamiento de alto nivel asociado. En la Figura 34 se muestra la clase que implementa los supresores de actividad. Los supresores extienden de la clase Comportamiento debiendo implementar los métodos abstractos definido en la clase base. El supresor mantiene internamente referencias a dos comportamientos que denominaremos comportamiento

supresor y comportamiento suprimido. Cuando el supresor esta en modo suprimir se hace pasar por el comportamiento supresor, en otro caso se hace pasar por el comportamiento suprimido.

Los métodos implementados en la clase Supresor son:

- darValor: retorna la acción a ejecutar. Si está suprimido retorna la acción indicada por el comportamiento supresor. En otro caso retorna la acción determinada por el comportamiento suprimido.
- darLargo: retorna la cantidad de sensores utilizados para la implementación del comportamiento.
- suprimir: este método es invocado por el comportamiento motivacional para suprimir el comportamiento de alto nivel asociado a él.
- activar: si está suprimido informa al comportamiento supresor que se activó. En otro caso informa al comportamiento suprimido.
- desactivar: si está suprimido informa al comportamiento supresor que se desactivó, en otro caso informa al comportamiento suprimido.
- finalizo: si está suprimido devuelve verdadero sí el comportamiento supresor finalizó. Si no está suprimido devuelve verdadero sí comportamiento suprimido finalizó. En el resto de los casos devuelve falso.
- reset: si está suprimido resetea el comportamiento supresor, en otro caso resetea el comportamiento suprimido.
- sensoryFeedback: si está suprimido devuelve el valor de *sensoryFeedback* retornado por el comportamiento supresor, en otro caso devuelve el valor de *sensoryFeedback* retornado por el comportamiento suprimido.

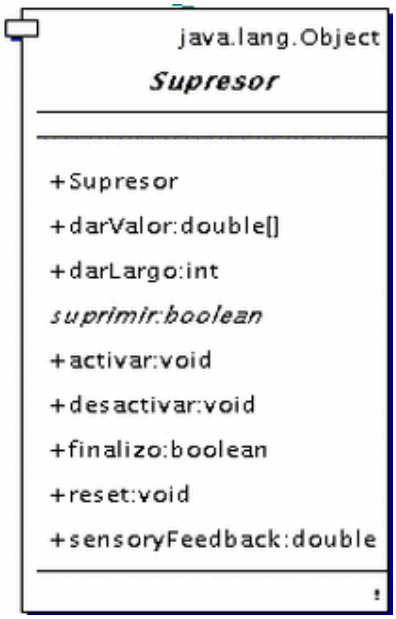


Figura 34. La clase Supresor.

B.5. NALLIANCE

Esta clase brinda la implementación de la arquitectura N-ALLIANCE. Para que un robot utilice N-ALLIANCE sólo debe crear una instancia de esta clase pasando como parámetros el identificador del robot, ρ, σ y el servicio de mensajería a utilizar.

Luego invocando al método addComportamiento se agregan los comportamientos de alto nivel que serán controlados por la arquitectura. Es importante notar que este método puede invocarse en cualquier momento permitiendo agregar capacidades al robot cuando sea necesario.

El método actuar es el encargado de actualizar las motivaciones de los distintos comportamientos de alto nivel y actualizar diversas estructuras dinámicas utilizadas por la clase UALLIANCE.

B.6. UNALLIANCE

Esta clase extiende a la clase NALLIANCE para permitir la actualización automática de parámetros. Para esto define el método abstracto updateParams que debe ser implementado por las clases que son responsables de las estrategias sobre NALLIANCE. En este método se llevan a cabo todas las modificaciones que se necesiten realizar sobre los parámetros de NALLIANCE de forma de implementar la estrategia deseada.

B.7. El paquete Presentación

Este paquete incluye todas las interfaces desarrolladas principalmente para presentar datos al usuario y permitir que este modifique parámetros de manera amigable.

B.7.1. La clase GControlManual

La clase control manual crea y ejecuta una instancia de NALLIANCE. Al crearla se le indican varios de los parámetros de NALLIANCE y la cantidad de comportamientos a controlar. . Su presentación es mostrada en la Figura 35. Desde la interfaz es posible controlar el sensoryFeedback y el taskTime de cada comportamiento. La misma presenta al usuario la tarea activa, la motivación para cada comportamiento, la categoría para cada tarea (entre paréntesis), los taskTime de los miembros del equipo, la cantidad de robots que participan de la misión y la cantidad de robots que están trabajando en una determinada tarea.

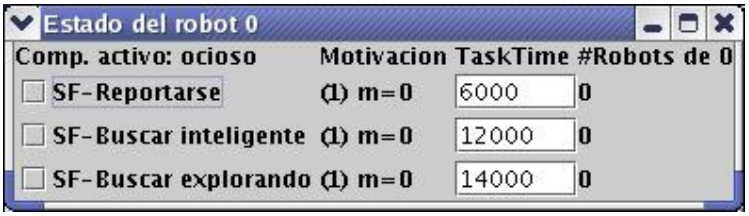


Figura 35. Interfaz gráfica manual con NUAlliance.

B.7.2. La clase ControlGrafico

La clase control gráfico presenta una interfaz gráfica que permite interactuar con el servidor del simulador YAKS. Esta interfaz se muestra en la Figura 36. Desde la misma se pueden controlar el motor izquierdo y el motor derecho utilizados para desplazarse, y los motores para controlar la pinza.

Figura 36. Interfaz gráfica para interactuar con el simulador YAKS.

B.7.3. La clase GQLearning

La clase GQLearning fue desarrollada en una etapa temprana del proyecto para conocer la actividad del aprendizaje del robot en una determinada tarea. En la Figura 37 se puede apreciar la interfaz gráfica presentada por la clase GQLearning. Desde la misma se puede modificar el modo de funcionamiento entre las siguientes opciones: aprender, aprender paso a paso, explotar o pausa. Se puede enviar un reset al comportamiento, reiniciar el contador de visitas a un par estado-acción, reiniciar los contadores de la interfaz, aumentar el tamaño de la tabla Q y leer o escribir una tabla Q desde o hacia un archivo. Además en todo momento la interfaz presenta la cantidad de iteraciones realizadas por el algoritmo, el tamaño actual de la tabla, la cantidad de recompensas recibidas, la cantidad de penalizaciones recibidas y la cantidad de refuerzos neutros.

Figura 37. Interfaz gráfica para el algoritmo QLearning.

B.8. Caso de Estudio

El caso de estudio elegido se implementó en la clase CasoDeEstudio cuyo código se muestra a continuación. Esta clase permite controlar un robot utilizando la arquitectura N-ALLIANCE con la estrategia CR/N-ALLIANCE.

```
public class CasoDeEstudio {
 /* RO_TIPO es el valor en milisegundos asignado al parámetro ro para todos
 * los robots
 */
 static long RO_TIPO = 1000;

 /* SIGMA_TIPO es el valor en milisegundos asignado al parámetro sigma para
 * todos los robots
 */
 static long SIGMA_TIPO = 5000;

 /* es el objeto utilizado como servicio de mensajería para todos los robots
 */
 static Mensajeria mensajeria=new SerNetMensajeria();

 /* es el objeto que implementa el protocolo para interactuar con el
 * simulador
 */
 static NetEntorno entorno = new NetEntorno();

 public static void main(String[] args) {
 /* recibe como parámetro el identificador de robot a controlar
 */
 if (args.length!=1) {
 System.out.println("uso: CasoDeEstudio idRobot");
 System.exit();
 }

 /* determino el identificador de robot
 */
 int idRobot=Integer.parseInt(args[0]);

 /* se definen las distintas coordenadas en el mundo
 */
 Coordenada posHome=new Coordenada(1000,1000);
 Coordenada posAlimIzq=new Coordenada(500,500);
 Coordenada posAlimDer=new Coordenada(1500,500);
 Coordenada posReport=new Coordenada (1200,1500);

 /* definición de los sensores necesarios para los comportamientos
 */
 Sensor sensoresRecIzq[]=new Sensor[15];
 Sensor sensoresRecDer[]=new Sensor[15];
 Sensor sensoresReport[]=new Sensor[5];
 Sensor sensoresKhe[]=new Sensor[KheperaK.CANT_SENSORES];
 for (int iterSens=0;iterSens<KheperaK.CANT_SENSORES;iterSens++)
 sensoresKhe[iterSens]=new Sensor();

 for (int iterSens=KheperaK.CANT_SENSORES;iterSens<sensoresRecIzq.length
```

```
;iterSens++) {
sensoresRecIzq[iterSens]=new Sensor();
sensoresRecDer[iterSens]=new Sensor();
}
for (int iterSens=0;iterSens<sensoresKhe.length;iterSens++) {
sensoresRecIzq[iterSens]=sensoresKhe[iterSens];
sensoresRecDer[iterSens]=sensoresKhe[iterSens];
}

sensoresReport[0]=new Sensor(posReport.getX());
sensoresReport[1]=new Sensor(posReport.getY());
sensoresReport[2]=sensoresKhe[KheperaK.CANT_SENSORES_REALES];
sensoresReport[3]=sensoresKhe[KheperaK.CANT_SENSORES_REALES+1];
sensoresReport[4]=sensoresKhe[KheperaK.CANT_SENSORES_REALES+2];

/* en las posiciones 0 a 10 vienen los sensores del khepera
* en las posiciones 11 y 12 viene la posicion del home
* en las posiciones 13 y 14 viene la posicion del alimento
*/
sensoresRecIzq[11].setValor(posHome.getX());
sensoresRecIzq[12].setValor(posHome.getY());
sensoresRecIzq[13].setValor(posAlimIzq.getX());
sensoresRecIzq[14].setValor(posAlimIzq.getY());

sensoresRecDer[11].setValor(posHome.getX());
sensoresRecDer[12].setValor(posHome.getY());
sensoresRecDer[13].setValor(posAlimDer.getX());
sensoresRecDer[14].setValor(posAlimDer.getY());

/* se crean los comportamientos de alto nivel
*/
EmpHome compEmpIzq=new EmpHome(sensoresRecIzq,KheperaK.CANT_ACTUADORES);
EmpHome compEmpDer=new EmpHome(sensoresRecDer,KheperaK.CANT_ACTUADORES);
Reportarse compReport=new Reportarse(sensoresReport,
KheperaK.CANT_ACTUADORES);
Braitenberg evitarObst = new Braitenberg(sensoresKhe,
KheperaK.CANT_ACTUADORES);

Braitenberg evitarObst2 = new Braitenberg(sensoresKhe,
KheperaK.CANT_ACTUADORES);

/* se definen los tiempos estimados para la ejecución de las tareas
*/
compReport.setTaskTime(45000);
compEmpIzq.setTaskTime(90000);
compEmpDer.setTaskTime(90000);

/* se crea el objeto necesario para controlar un robot del simulador
*/
NetRobot robot=new NetRobot(idRobot);
RobotReal robotReal;

/* se crea la arquitectura de comportamiento utilizando los
* comportamientos de alto nivel y los supresores
*/
```

```
SupresorLlave supresores[]=new SupresorLlave [3];
SupresorSensoryFeedback supReportEvitarObst;
supresores[0]=new SupresorLlave(evitarObst,compEmpIzq);
supresores[1]=new SupresorLlave(supresores[0],compEmpDer);
supReportEvitarObst=new SupresorSensoryFeedback(compReport,evitarObst2);
supresores[2]=new SupresorLlave(supresores[1],supReportEvitarObst);

/* se crea un objeto que interactua entre el simulador y la arquitectura
* de comportamientos
*/
robotReal=new RobotReal(robot,supresores[2],sensoresKhe);

/* se obtiene una referencia a los sensores que serán actualizados con los
* datos recibidos del simulador
*/
sensoresKhe=robotReal.getSensores();

/* se crea una instancia de la clase CantidadRobots que implementa la
* estrategia CR/N-ALLIANCE.
*/
CantidadRobots ALLIANCE = new CantidadRobots(idRobot, RO_TIPO,
TAU_TIPO,mensajeria);
/* se crean los comportamientos motivacionales
*/
CMotivacional comMotReportarse=new CMotivacional(ALLIANCE,
Comportamientos.REPORTARSE, compReport,supresores[2]);
CMotivacional comMotEmpIzq=new CMotivacional(ALLIANCE,
Comportamientos.RECOLECTAR_IZQUIERDA, compEmpIzq,supresores[0]);
CMotivacional comMotEmpDer=new CMotivacional(ALLIANCE,
Comportamientos.RECOLECTAR_DERECHA, compEmpDer,supresores[1]);

/* ajusta los valores de minDelay y maxDelay definidos en L-ALLIANCE.
*/
ALLIANCE.setMinDelayUnit(5000);
ALLIANCE.setMaxDelayUnit(10000);

/* agrega los comportamientos a la arquitectura
*/
ALLIANCE.addComportamiento(comMotReportarse);
ALLIANCE.addComportamiento(comMotEmpDer);
ALLIANCE.addComportamiento(comMotEmpIzq);

/* por siempre ejecuto la misión asignada al robot
*/
while (true) {
/* actualizo los parámetros de la arquitectura ALLIANCE
*/
ALLIANCE.actuar();

/* actualizo los sensores e interactuo con el mundo
*/
robotReal.actualizar();
}
}
}
```

Apéndice C. Robots Actuales

C.1. Introducción

En este apéndice se realiza una pequeña presentación de los robots AIBO, SDR-4X y ASIMO que son claros representantes del estado actual de la tecnología en el diseño de robots. Primero se presenta al robot AIBO cuyo objetivo principal es el entretenimiento aun cuando éste se usó para la investigación. Luego se presentan los robots humanoides SDR-4X y ASIMO que están orientados a entretenimiento y a ayudar a las personas en tareas del mundo real, respectivamente.

C.2. El Robot AIBO

Este robot entra en la categoría entretenimiento. Es simplemente un robot perro, que como tal obedece las ordenes de su amo si está de buen humor.

Figura 38. El robot AIBO.

Este robot es construido por la empresa SONY y actualmente se encuentra a la venta la segunda generación la cual presenta mayor habilidad para expresar sus emociones. Este modelo incluye la capacidad de darle un nombre al robot para luego llamarlo por éste, el reconocimiento de palabras simples y la capacidad de tomar fotografías cuando se le ordena (seguro que su mascota no puede hacerlo).

Cuando se instala en AIBO el software adecuado éste actúa de manera completamente autónoma y puede tomar decisiones acerca de sus acciones y comportamiento. AIBO presenta instintos como ir a buscar su juguete preferido, mostrar deseos de jugar con las personas, explorar, descansar y alimentarse. AIBO expresa mediante luces, sonidos y gestos las siguientes emociones: alegría, tristeza, enojo, sorpresa, miedo y descontento.

AIBO puede ponerse un poco travieso y no hacer caso de las ordenes que se le indican cuando está de mal humor o desea ir a dormir.

La tecnología utilizada en AIBO le permite actuar de forma autónoma, aprender y madurar.

En la Figura 38 se muestra la última versión del robot AIBO y en la Tabla 15 se presentan las características principales del mismo.

Tabla 15. Características técnicas del robot AIBO.

Elemento	Descripción
Procesador	RISC de 64 bits
Memoria principal	32MB SDRAM
Actuadores	4 piernas con 3 grados de libertad cada una. 4 grados de libertad en la cabeza. 2 grados de libertad en la cola. Parlante.
Sensores	2 Micrófonos. 1 cámara de video. 1 sensor de temperatura. 5 sensores de contacto. Sensor de aceleración. Sensor de vibración.
Energía	Batería de NiCd recargable
Autonomía	1.5 hs.
Tamaño	152x296x278 mm
Peso	Aproximadamente 1.5 Kg.

C.3. El Robot Asimo

El robot humanoide ASIMO nace en el año 2000 a partir del desarrollo de su predecesor el robot P3. Este fue desarrollado por la empresa HONDA con la idea de crear un robot que pueda moverse en la sociedad ayudando a sus compañeros humanos.

Al momento de decidir el tamaño y la apariencia del robot los diseñadores pensaron en hacerlo amigable y práctico. Las dimensiones elegidas le permiten a ASIMO moverse en el entorno humano y realizar tareas útiles, como ser prender luces, abrir puertas y trabajar en las mesas usados por los humanos.

La tecnología denominada i-WALK utilizada en ASIMO le permite caminar de forma continua mientras cambia de dirección, su predecesor se detenía antes de realizar el cambio de dirección. De esta forma ASIMO logra movimientos más suaves y naturales (humanos).

ASIMO puede controlarse utilizando una computadora o mediante un dispositivo portátil de control.

En la Figura 39 se muestra la última versión del robot ASIMO y en la Tabla 16 se presentan las características principales del mismo.

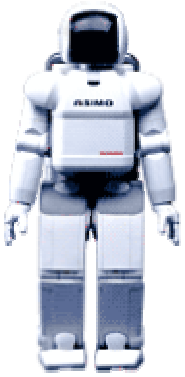


Figura 39. El robot ASIMO

Tabla 16. Características técnicas del robot ASIMO.

Elemento	Descripción
Peso	43Kg
Velocidad	0 a 1.6 km/h
Actuadores	Servomotor
Sensores	Giroscopio. Aceleración. 2 cámaras.
Grados de libertad	Cuello 2. Hombros 3. Codo 1. Muñeca 1. Mano 1 Cadera 3. Rodilla 1. Tobillo 2.

C.4. El Robot SDR-4X II

El robot bípedo SDR-4X (Sony Dream Robot) fue desarrollado por SONY en marzo de 2002, siendo el entretenimiento el principal mercado al cual apunta esta empresa. En 2003 aparece el modelo SDR-4X II, que mejora la movilidad y las características de reconocimiento y comunicación respecto de su predecesor.

Está capacitado para moverse en tiempo real entre obstáculos y superficies irregulares, e incluso puede mantener la postura frente a presiones externas. Para esto utiliza principalmente dos cámaras CCD para determinar la distancia entre él y los objetos del entorno, y calcular para evitar los objetos percibidos. En caso de sufrir una caída el robot adopta inmediatamente la postura que minimiza sus daños.

Este robot puede reconocer personas a partir de la imagen capturada por la cámara color. Es capaz de reconocer la dirección de una fuente de sonido, más aún puede reconocer la voz de un individuo, lo que está diciendo e interactuar con él utilizando su parlante.

En contrate con el robot ASIMO, las dimensiones del robot SDR no lo hacen muy útil para trabajar en conjunto con los humanos.

En la Figura 40 se muestra la última versión del robot SDR y en la Tabla 17 se presentan las características principales del mismo.

Figura 40. El robot SDR-4X II.

Tabla 17. Características técnicas del robot SDR-4X II.

Elemento	Descripción
CPU	Tres CPUs 64 bit RISC
Memoria	192MB DRAM
Sistema Operativo	Aperios
Grados de libertad	Cuello 4 Cuerpo 2 Brazo 5. Pierna 6
Sensores	Infrarrojo, aceleración, giroscopio, contacto, cámara, temperatura y micrófono
Peso	7 kg (incluida la batería)
Dimensiones	580x270x190 mm
Velocidad	0.36 km/h en superficies irregulares 1.2 km/h en superficies lisas