Sistema multiagentes para el comando oral de un robot móvil

Oscar Pérez Márquez [1], Luis Villaseñor Pineda [2]

[1] Benemérita Universidad Autónoma de Puebla
Av. San Claudio y 14 Sur
Puebla, México.
osmarp79@hotmail.com

[2] Laboratorio de Tecnologías del Lenguaje
Instituto Nacional de Astrofísica, Óptica y Electrónica
Calle Luis Enrique Erro #1.
Sta. María Tonanzintla, Pue., México
villasen@inaoep.mx

Resumen. El presente trabajo presenta el diseño y los resultados de la implementación de una arquitectura multiagentes para el comando oral de un robot móvil. La idea central fue la búsqueda de una plataforma flexible capaz de integrar todos los elementos posibles para poder brindar una solución satisfactoria al comando oral. Por supuesto esto no es una tarea fácil, la máquina debe escuchar la orden, comprenderla y ejecutarla. A pesar de que estas tareas son extremadamente complejas, en situaciones con contextos cerrados y bien definidos es posible llegar a tener éxito. Este es el caso del comando de un robot móvil. Actualmente la plataforma está completa y funciona correctamente. En esta primera etapa se han integrado diversos procesos hasta obtener una plataforma básica y funcional. Sin embargo, la plataforma brinda el medio para incluir agentes especializados o intercambiar los existentes.

1 Introducción

La capacidad de entender y usar el lenguaje natural es un tema que la ciencia ficción da por hecho. Por ejemplo, en la película "2001 Odisea del Espacio" la computadora HAL 9000 era capaz de interactuar con los seres humanos por medio del lenguaj natural sin ningún problema. Por supuesto, la distancia entre la ficción y la realidad este tema es aún enorme. Sin embargo, la idea de interactuar con una máquina usand el lenguaje, en particular el lenguaje oral, nos brindaría una enorme ventaja. El simpl hecho de no necesitar aprender un medio artificial de comunicación sería suficiente. Por supuesto esto no es una tarea fácil, la máquina debe escuchar la orden, comprenderla y ejecutarla. A pesar de que estas tareas son extremadament complejas, en situaciones con contextos cerrados y bien definidos es posible llegar tener éxito. Este es el caso del comando de un robot móvil.

El presente trabajo presenta el diseño y los resultados de la implementación de arquitectura multiagentes para el comando oral de un robot móvil. La idea central

A. Gelbukh, M. Hernández Cruz (Eds.) Avances en la ciencia de la computación en México, CORE-2003, pp. 126–134, 2003.

la búsqueda de una plataforma flexible capaz de integrar todos los elementos posibles para poder brindar una solución satisfactoria. Actualmente la plataforma está lista y funciona correctamente. En esta primera etapa se han integrado diversos procesos hasta obtener una plataforma básica y funcional. Sin embargo, la plataforma brinda el medio para incluir agentes especializados o intercambiar los existentes. Actualmente la plataforma integra aplicaciones completas como son el reconocedor de voz –en este caso se utiliza un sistema comercial– y el subsistema de control de movimientos de bajo nivel del robot –un sistema propio del robot sobre el cual trabajamos–. Por otro lado, se desarrolló un sistema para el procesamiento sintáctico-semántico de la orden enunciada por el usuario para poder adecuarla al contexto del robot y así conseguir su realización.

La siguiente sección explica brevemente qué es un agente y qué es un sistema multiagentes. Ella termina con la presentación del *toolkit* usado en nuestro sistema. La tercera sección presenta la arquitectura de nuestro sistema y describe los agentes principales y sus funciones específicas. Finalmente, se describen los resultados obtenidos hasta ahora y el trabajo futuro a realizar.

2 ¿Qué es un agente?

Actualmente no hay una definicion aceptada globalmente de un agente de software [1,2]. En vez de esto, los agentes se definen con base en la línea de investigación y/o aplicación donde se desarrollan. Sin embargo, la siguiente definición es suficiente para discutir algunos aspectos de los agentes.

> "Un agente de software es una entidad computacional que desempeña de forma automática tareas delegadas por el usuario" [1]

Esta definición implica una metáfora con un asistente personal, donde el agente realiza tareas en vez del usuario. Ejemplos de estos agentes son: agentes de filtrado de correo electrónico, agentes de recuperación de información y agentes de automatización de tareas de escritorio.

Una definición más formal dice:

> "un agente es todo aquello que puede considerarse que percibe su ambiente mediante sensores y que responde o actúa en tal ambiente por medio de efectores; por otro lado un agente racional deberá emprender todas aquellas acciones que favorezcan obtener el máximo de su medida de rendimiento, basándose en las evidencias aportadas por la secuencia de percepciones y en todo el conocimiento incorporado en tal agente" [3]

Otro aspecto por definir es agente *inteligente*. Según [4] podríamos decir que es aquel agente capaz de realizar acciones autónomas de forma ***flexible***, es decir, cumpliendo con estas tres características:

- Reactividad: se refiere al hecho de que un agente debe poder sensar el estado del ambiente dentro del cual se encuentra inmerso y en función de esto, actuar,

respondiendo de manera adecuada a cambios producidos en el mismo. efectos producidos pueden modificar el estado de su entorno.

- pro-actividad: se podrá tomar la iniciativa en las decisiones, tener comportamiento basado en objetivos.
- Habilidad social: mostrar capacidad para interactuar con otros agentes incluso con humanos)

Lo ideal en un agente inteligente es lograr un compromiso entre comportamien basado en objetivos y conducta reactiva. La razón es que un agente deberá intent cumplir sus objetivos sistemáticamente, pero deberá también darse cuenta de cuán el procedimiento que actualmente sigue ya no es el apropiado, decidiendo en momento qué nueva acción llevar a cabo.

2.1 Arquitecturas concretas para agentes inteligentes

Consideraremos cuatro clases de agentes, dependiendo de cómo toman las decision [4]:

- Basados en lógica, a través de deducciones lógicas.
- Reactivos, que transforman directamente situaciones en acciones.
- Creencia – deseo – intención, que dependen de la manipulación de estructuras datos que representan las creencias, los deseos y las intenciones del agente.
- Arquitecturas en capas, en las que se realiza a través de varias capas de softwa cada una de las cuales representa un nivel de abstracción diferente sobre el entorno.

Frecuentemente los agentes no son desarrollados de forma aislada, sino con partes de un sistema complejo poblado por una diversidad de agentes, los sistem Multiagente (MAS) [9]. Los MAS son sistemas computacionales en los cuales vari agentes semi-autónomos interactúan o trabajan juntos para desempeñar un conjun de tareas, o para satisfacer un conjunto de metas.

2.2 Agentes inteligentes Jack

Los agentes inteligentes están siendo usados para modelar conductas simpl racionales en un amplio rango de aplicaciones distribuidas. En particular, arquitecturas multiagentes basadas en el modelo BDI han sido usadas exitosamente situaciones donde el modelo de razonamiento humano y la conducta en equipo necesarios.

La compañía Agent Oriented Software (AOS) que esta en Melbourne, Australia construido los agentes inteligentes Jack, que es un *toolkit* en Java para desarroll sistemas multiagentes. Jack utiliza agentes inteligentes basados en la conducta razonamiento BDI [6].

JACK es un entorno de desarrollo orientado a agentes. Esta completamente integrado con el lenguaje de programación Java. JACK consiste de los siguientes componentes claves:

- El lenguaje de agente de Jack. JAL(Jack Agent Language) es usado para desarrollar sistemas basados en agentes integrando nuevas clases, interfaces y métodos a Java.
- El compilador de JACK El compilador preprocesa los archivos JAL y los convierte a Java. Este código de Java puede ser entonces compilado en una máquina virtual de Java para correr en el sistema.
- El Kernel de JACK Provee un conjunto de clases que dan a los programas JAL su funcionalidad orientada a agentes.

Un agente de Jack es un componente de software que puede exhibir conducta de razonamiento bajo estimulación reactiva (manejando eventos) y proactiva.
Cada agente de Jack tiene:

- Creencias (*beliefs*) acerca del mundo (su conjunto de datos).
- Eventos a los cuales responder.
- Metas que desea alcanzar.
- Planes que describen como manejar las metas o eventos que podrían surgir

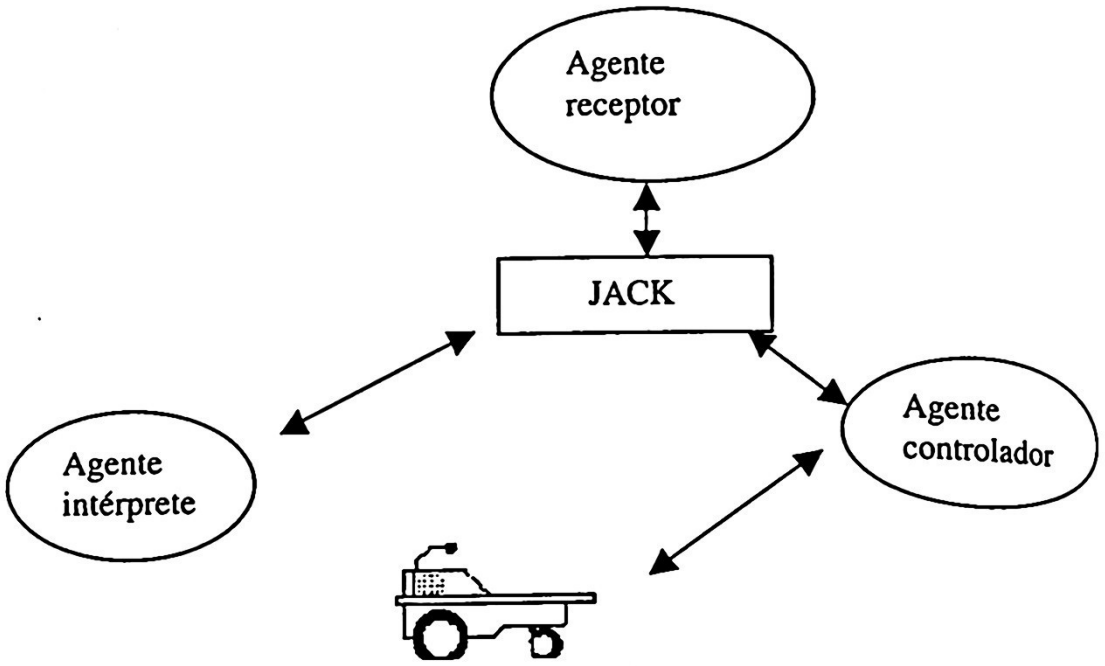
La forma en la que funciona un agente Jack es la siguiente: al momento en que un agente de JACK es instanciado en un sistema espera a que sea dada una meta o experimente un evento al cual deba responder, una vez que se recibe un evento o meta el agente inicia una actividad para manejarlo, esto lo hace buscando que plan es el más adecuado para manejar dicho evento y entonces ejecuta el plan. La ejecución del plan podría involucrar interacción con las relaciones de creencia de un agente u otra estructura de datos de Java. Los planes pueden llevarse a cabo o fallar, bajo ciertas circunstancias, si el plan falla el agente pudiese intentar otro plan.
Dentro de un proceso se puede tener múltiples agentes o también se puede tener agentes en diferentes procesos y sobre diferentes máquinas comunicándose uno con otro.

3 Arquitectura del sistema

La implementación de nuestro sistema se realizó utilizando la plataforma multiagentes Jack, la cual nos permite manejar un ambiente distribuido en donde se pueden tener un grupo de agentes trabajando en procesos que se encuentran en diferentes máquinas. Esto es de particular interés en nuestro caso, ya que existen procesos muy costosos y bastará con repartir adecuadamente la carga de cómputo para mejorar el rendimiento de nuestra aplicación.
El sistema esta dividido en tres procesos que son: el receptor de la orden, el intérprete de órdenes y el controlador de movimientos. Cada uno de estos procesos es llevado a cabo por un agente, todos ellos comunicados a través de la plataforma multiagentes. Cada agente registra sus capacidades de manera que pueda recibir peticiones de los otros. Por supuesto, los agentes pueden ejecutarse dentro de una misma máquina o en máquinas distintas, el máximo número de máquinas que se

pueden utilizar para la ejecución de nuestro sistema debe ser el mismo que el número de agentes utilizados.

El esquema del sistema es el siguiente:



Las siguientes secciones detallan cada uno de estos procesos.

3.1 Receptor de la orden

La función principal que realiza este agente es la captura de la orden pronunciada el usuario y su transferencia al agente intérprete. Para el reconocimiento de voz utilizó el sistema comercial Dragon Naturally Speaking. Este se encarga transformar la señal acústica en texto. Para mejorar el rendimiento del reconocedor un proceso de entrenamiento debe realizarse. El entrenamiento consiste en leer un texto durante aproximadamente 15 minutos.

El agente receptor presenta una interfaz al usuario en la cual el agente recibe peticiones y muestra información acerca de lo que está haciendo el robot. Será través de esta interfaz que el agente recibe el texto *reconocido* por Dragon y una obtenida la orden esta es transmitida al siguiente proceso.

La GUI contiene componentes que nos permiten visualizar información acerca Robot como la velocidad a la que se mueve, las coordenadas de donde se encuentra, estado de la comunicación, etc. La figura 1 muestra esta interfaz. Además nos permi elegir entre varias opciones para el manejo del sistema como son: modo de entrac (voz o teclado), elegir si se va a trabajar con el robot físico o con el simulador y permite establecer la conexión con los demás agentes.

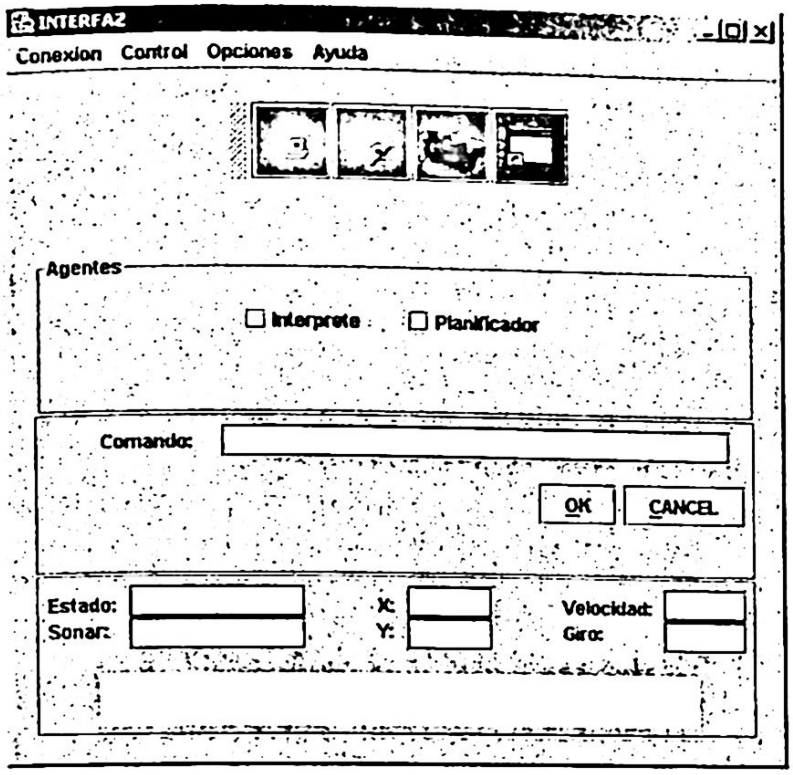


Fig. 1. Interfaz para la recepción de la orden.

El agente Receptor realiza otras tareas, como es la de comunicarse con el agente controlador para conectarse al robot, obtener datos del robot como la velocidad a la que avanza, el estado en el que se encuentra y las coordenadas que tiene robot actualmente para así poder desplegarlas en la interfaz de usuario.

3.2 Intérprete de órdenes

El intérprete se encarga de transformar el texto de la orden, expresado en lenguaje natural, a comandos expresados en el lenguaje de control del robot.El intérprete es un agente en Java que encapsula un motor de Prolog. Así el intérprete está propiamente programado en Prolog y usa el formalismo de DCGs (Definite Clause Grammars) para representar las estructuras gramaticales manejadas [10]. Al terminar la transformación de la orden, si se trata de una orden válida, se entregará el comando correspondiente al controlador de movimientos.

Las DCGs facilitan la programación de gramáticas libres de contexto. Una gramática en notación DCG es directamente ejecutada por Prolog. Además, la notación DCG permite incorporar la semántica del lenguaje en las reglas de producción de tal modo que el significado de un enunciado pueda ser manejado al mismo tiempo que la sintaxis.

Nuestra gramática depende en gran medida de las tareas que pueda hacer el robot, y por el momento se tienen frases que involucran movimientos simples del robot. Las frases que utilizamos en nuestra gramática prácticamente las podemos clasificar en tres tipos:

Órdenes simples: Estas órdenes están compuestas por un verbo en imperativo y algún adverbio. Por ejemplo: camina, avanza, retrocede, alto, camina más rápido, vete un poco más despacio.

Ordenes compuestas: Estas órdenes están formadas por una frase verbal, una frase nominal indicando la distancia o grados de giro y una frase preposicional indicando dirección. Por ejemplo, camina dos metros hacia adelante, camina rápido 4 metros hacia delante, camina hacia atrás, gira a la derecha 90 grados, gira 180 grados hacia derecha, camina hacia la izquierda. Es también posible construir frases donde la frase verbal se da por sobreentendida (elipsis), por ejemplo: hacia adelante un metro, metros hacia delante, a la derecha 90 grados, 90 grados a la izquierda.

Ordenes para manipulación de la cámara: Dado que el robot esta equipado con una cámara también es posible dirigirla a la una dirección específica. Por ejemplo, mira hacia la derecha, mira hacia la izquierda, mira hacia el frente.

3.3 Controlador de movimientos

La principal función del controlador es ejecutar el comando que recibe del intérprete. Además tiene otras funciones como envíar información al agente receptor de la orden sobre el estado del robot, velocidad, las coordenadas en las que se encuentra y distancia a la cual se encuentra el objeto más cercano. Los comandos que por ahora puede ejecutar el robot son simples como por ejemplo de giro, de traslación, modificación de velocidad.

Este agente está a cargo de salvaguardar al robot. Cuando el robot está movimiento este agente toma las lecturas de los sonares y en caso de detectar obstáculo cercano envía al robot la orden de detenerse para evitar así cualquier choque.

El agente controlador es un agente en java que encapsula el software de comando del robot Pioneer el cual está programado en C. Esto se hizo a través de métodos nativos. Un método nativo es un método de java cuya implementación está escrita otro lenguaje de programación. Los métodos nativos se integran a Java utilizando JNI (Java Native Interface). JNI es una interfaz de programación nativa y permite que código de Java que corre dentro de la máquina virtual, interopere con aplicaciones librerías escritas en otro lenguaje de programación, tales como C, C++ ensamblador. [8]

Los métodos nativos que se implementan en este proyecto están escritos en lenguaje de programación C, además de usar las instrucciones que controlan al robot, las cuales están contenidas en la librería PAI y la librería de Saphira. Saphira es entorno de desarrollo de aplicaciones robóticas desarrollado en el centro Inteligencia Artificial del Stanford Research Institute (SRI) bajo la dirección de Kurt Konolige [7]. Este programa nos permite crear nuestras propias funciones lenguaje C y hacer uso de los comandos de control directo que se encuentran definidos en su librería. Saphira incluye un software de simulación del robot, el cual utilizamos para probar nuestro sistema antes de hacerlo con el robot físico.

PAI es una librería de funciones y definiciones en C para controlar un robot móvil Pioneer. Tiene acceso a todas las funciones del robot. PAI hace uso de las funciones

de control de bajo nivel de Saphira. Esto provee la opción de usar algunas funciones de Saphira en un programa.

En el agente controlador se declaran los siguientes métodos nativos:

```
public native int Estado();
public native int ObtenerX();
public native int ObtenerY();
public native int SonarDistancia();
public native int SonarFrente();
public native int ConectarSimulador();
public native int ConectarRobot();
public native void Move(String n);
public native void Turn(String n);
public native void Stop();
public native void Desconectar();
public native void setVelocidad(String n);
public native void InicializaCamara();
public native void Pan(String n);
public native void Tilt(String n);
public native void PanTilt(String p,String t);
public native void Zoom(String n);
public native void Sonido();
```

Estas funciones se compilaron en un DLL para así ser accesibles desde el agente controlador.

4 Resultados obtenidos hasta ahora

Actualmente, el sistema se encuentra funcionando y ha sido probado en condiciones reales. Para ello, diferentes usuarios han sido invitados a usar el sistema y han comandado oralmente el robot. Diferentes configuraciones han sido probadas: los tres agentes en una sola máquina así como un agente por máquina. En todos los casos, los resultados han sido satisfactorios.

Por otro lado, nos encontramos en la fase de estudio de la cobertura de nuestra gramática. Para ello se han diseñado una serie de experimentos sencillos donde se pondrán a varios usuarios a comandar el robot. Cada sesión será grabada y se deberá establecer si aún hay formas no previstas de comandar oralmente al robot.

5 Trabajos futuros

Ya con la plataforma funcionando se tienen tres proyectos a futuro:

- Transmisión de la señal de video de la cámara del robot a la terminal de comando. Se desea poder ver desde el puesto de control la imagen percibida con la cámara del robot. Gracias al comando vocal de la cámara el robot podrá mostrarnos su entorno.
- Durante las sesiones de prueba con el robot, se descubrió que existe una diferencia entre el resultado final deseado y el resultado final obtenido. Esto se

presenta dadas las imperfecciones de la superficie, pequeñas diferencias entre el arranque de los motores, etc. Esto provoca que la posición final esperada no coincida con la real. Un trabajo a futuro es la consideración de mapas de localización para reposicionamiento.

- Comando multimodal del robot, voz y ademanes. En convenio con otros centros de investigación será posible coordinar esfuerzos e intentar la integración de un sistema que sea capaz de fusionar la voz y el reconocimiento de ademanes para comandar el robot.

Agradecimientos

Los autores agradecen al CONACYT por el apoyo financiero para la realización de este trabajo (No. Ref. 31128A). En particular, la beca de tesis de licenciatura recibida por el primer autor. Los autores también agradecen el apoyo del Laboratorio Tecnologías del Lenguaje y el Laboratorio de Robótica Móvil del INAOE.

Referencias

1. Alper Caglayan and Colin Harrison, Agent Sourcebook, Wiley Computer Publishing, 1997
2. Jörg P. Muller ,The Design of Intelligent Agents, A Layered Approach , Springer-Verlag Berling Heidelberg, 1996
3. Stuart Russell, Peter Norving, Inteligencia Artificial, Un enfoque moderno, Ed. Prentice Hall, 1996
4. Weiss, Gerard. Multiagent Systems: A modern Approach to Distributed Artificial Intelligence, Edited by Gerhard Weiss. MIT Press, 1999.
5. Manuel Pérez, Sistema Multiagentes para creación y mantenimiento de hipertexto a partir de documentos en español, Tesis de Maestría en Ciencias Computacionales, INAOE, Puebla México, Febrero 2003.
6. Nick Howden, Ralph Rönnquist, Andrew Hodgson, Andrew Lucas, JACK Intelligent Agents-Summary of an Agent Infrastructure, Agent Oriented Software Pty. Ltd.
7. Miguel Cazorla, Domingo Gallardo, Introducción a Saphira, Universidad de Alicante, Octubre 1998.
8. Beth Stearns, Java Native Interface, Sun Microsystems, 1995-2002.
9. Michael N. Huns and Munindar P. Singh, Readings in Agents, Morgan Kaufmann Inc, 1998.
10. Manual de referencia de SICStus Prolog 3.8.7, Octubre 2001.