UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Posgrado en Ciencia e Ingeniería de la Computación

PLANEACIÓN DE ACCIONES PARA UN ROBOT DE SERVICIO A TRAVÉS DE COMANDOS DE VOZ

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIAS (COMPUTACIÓN)

PRESENTA:
CRUZ ESTRADA JULIO CESAR

DIRECTOR DE TESIS:
DR. JESÚS SAVAGE CARMONA
FACULTAD DE INGENIERÍA

CIUDAD UNIVERSITARIA, CD.MX ENERO 2017

AGRADECIMIENTOS

Esta tesis marca la culminación de mi etapa como estudiante de la maestría en ciencia e ingeniería de la computación, han sido dos años de estudio, esfuerzo y sufrimiento que se han visto convertidos en la obtención de uno de mis más grandes logros en la vida. Sin embargo este camino no lo he recorrido solo, siempre tuve el apoyo de personas increíbles a las cuales quiero agradecer y dedicar la presente tesis.

A mis padres Leonor Estrada y Miguel Cruz quiero que sepan que no existirá forma alguna de agradecerles una vida de sacrificios, esfuerzos y amor, quiero que sientan que el objetivo alcanzado también es de ustedes y que la fuerza que me ayudo a conseguirlo fue su gran apoyo.

A mis hermanas Claudia y Gabriela agradesco su apoyo, comprensión y confianza esperando que comprendan que mis logros son también suyos e inspirados en ustedes.

A Hugo, David, Arely, Beto, Charly, Angel, Crispín, Alan, Emmanuel, Lupita, Yes, Erika, Mario y Reynaldo por su valiosa amistad y el apoyo que recibí de ustedes muchas gracias.

De manera muy especial quiero agradecer al Dr Jesús Savage Carmona, director de tesis; a la maestra Norma Elva Chávez Rodríguez, que en este trayecto de mi vida influyeron con sus conocimientos, consejos y experiencias para formarme en una mejor persona. Para ustedes mi gratitud y respeto.

A todo el equipo de Bio-robótica. En este par de años ha sido un placer trabajar con ustedes. Gracias por el apoyo y por su amistad.

Al IIMAS y sobre todo a la Universidad Nacional Autónoma de México porque puedo decir con orgullo que pertenecí a esta institución y que en sus aulas adquirí los valores que la definen permitiendo así una formación integral en mí.

Agradezco también a CONACYT por la beca otorgada durante la realización de este posgrado.

Finalmente agradezco a la DGAPA-UNAM por el apoyo proporcionado para la realización de esta tesis a través del proyecto PAPIIT IG100915 "Desarrollo de técnicas de la robótica aplicadas a las artes escénicas y visuales.

Hago de este un triunfo y quiero compartirlo por siempre con ustedes.

Julio Cesar Cruz Estrada

RESUMEN

Esta tesis presentó el diseño e implementación de un planeador de acciones para un robot de servicio, al cual se le hacieron peticiones por medio de comandos de voz. En otras palabras se desarrolló un sistema basado en conocimiento que interpretara oraciones en lenguaje natural y generará una secuencia de acciones de alto nivel, las cuales dotaron al robot de comportamientos y conductas, parecidas a las del ser humano, que le permitieron manipular la representación del mundo donde interactuó.

El sistema constó de 2 partes principales:

1. **El mecanismo de comunicación**: Donde se definió la estructura gramatical de los comandos, se realizó el análisis sintáctico y finalmente la interpretación semántica del comando. El resultado final fue una expresión formal que representa una secuencia de acciones de alto nivel que el robot debe ejecutar.

2. **El mecanismo de ejecución**: Con el resultado de la interpretación del comando, el planeador, a base de reglas y el uso de inferencias, determinó que serie de tareas y subtareas (acciones de menor nivel) ejecutar, para que el robot completará las peticiones que se le hicieron.

Finalmente el planeador de acciones fue usado en la competencia internacional Robocup@Home al integrarlo al robot Justina, desarrollado en el laboratorio de Bio-Robótica en el Posgrado de Ingeniería, UNAM. En particular colaboré en la prueba General Purpose Service Robot en la edición 2016 de la competencia Robocup@Home, donde se obtuvo el décimo lugar.

ÍNDICE

▫

ÍNDICE DE FIGURAS

CAPÍTULO 1

INTRODUCCIÓN

Robots que puedan ser usados como ayudantes eficientes en la industria o incluso en ambientes domésticos son un objetivo de alta prioridad en investigaciones actuales. Esto produjo una vertiente en la cual se persigue la creación de sistemas cada vez más independientes y que realicen tareas con una complejidad cada vez mayor, hasta llegar a crear sistemas autónomos que ayuden a llevar a cabo algunas actividades que cualquier persona podría hacer; tal es el caso de los robots de servicio.

Se espera que un robot de servicio de propósito general sea capaz de realizar un conjunto de actividades de alto nivel con la menor ayuda posible en un ambiente doméstico o similar, donde interactúe con personas y objetos cotidianos; estas actividades involucran comportamientos básicos como navegar, reconocer y manipular objetos, así como reconocer e interactuar con personas.

El desarrollo de la robótica en el ámbito de la asistencia en la vida cotidiana representa muchos retos aún por superar, inherentes a un ambiente dinámico y poco controlado, a la complejidad en el manejo de los sensores y a las acciones que debe llevar a cabo, así como del nivel de abstracción que requiere la tarea. Estas dificultades usualmente se aprecian cuando el robot actúa de una forma que no corresponde con lo que el usuario espera que haga. Por esta razón es importante modelar la estructura de la tarea de una forma que permita manejar de manera eficaz la mayor cantidad de situaciones que se puedan presentar de este tipo.

La “planeción de acciones” en el campo de la robótica ha sido investigada durante varios años hasta la actualidad. Sin embargo existen pocas soluciones que son lo suficientemente eficientes para permitir una re-planeación en ambientes dinámicos con la gran complejidad que conlleva realizar tareas

domésticas. Por lo que la planeación se realiza offline y la ejecución está acompañada de comportamientos reactivos para compensar pequeñas perturbaciones.

La planeación de acciones en general hace referencia a la selección y ordenamiento de acciones dentro de un grupo de posibilidades con la finalidad de llevar a cabo una tarea más compleja o alcanzar un objetivo de manera eficiente. En el campo de robótica, consiste en dirigir las acciones del robot de tal forma que realice lo que se le pide o se espera de él. Para lograr esto debe tomar información del medio a través de sus sensores.

Hipotesis

La planeación de acciones es de gran importancia cuando se desarrolla en un ambiente dinámico y poco controlado. Por esa razón en este trabajo de tesis se desea contestar a la pregunta: ¿La construcción de un motor de planeación para robots de servicio basado en un sistema experto, con el cual se puede modelar conocimiento humano podrá proveer al robot comportamientos que le permitirán manejar con resultados positivos peticiones del usuario a través de comandos de voz y, así mismo, alcanzar una forma de interacción humano-robot más natural?.

En el presente trabajo de tesis cabe destacar la importancia de obtener un método especializado en la planeación de acciones. Es relevante, ya que en la literatura existen pocas propuestas sobre este tema y es un tema de estudio relativamente nuevo en el laboratorio de Bio-Robótica que no dispone de un sistema de planeación de acciones. El desarrollo de este trabajo contribuirá en el campo de los robots de servicio, y permitiendo que el laboratorio de Bio-Robótica alcance el objetivo de tener un robot con más flexibilidad o libertad, para moverse en un ambiente real y para tener una interacción mas natural con los humanos.

Para el desarrollo de esta tesis se pretende realizar la simulación de la planeación de acciones del robot Justina sobre el ambiente Robot Operating System (ROS). ROS es un framework utilizado en el desarrollo de software

para robots. Es una colección de herramientas, librerias y convenciones para simplificar las tareas de crear comportamientos del robot robustos y complejos a traves de una amplia variedad de plataformas empleadas en robótica.

ROS cuenta con una variedad de paquetes para distintos usos e implementaciones, uno de ellos es actionlib, del cual se hara uso para el desarrollo de esta tesis. El paquete actionlib brinda una interfaz estandarizada para la interaccion con ciertas tareas, por ejemplo: navegar hacia un objetivo, utilizar un scanner laser, detección de objetos, etc. Además actionlib provee herramientas para crear servidores que ejecuten objetivos de tiempo de ejecución largo, asi como una interfaz de cliente con el objetivo de enviar peticiones al servidor.

Motivación

Este trabajo se realizó en el contexto del Laboratorio de Bio-Robótica de la Facultad de Ingeniería de la UNAM, dirigido por el Dr. Jesús Savage Carmona, donde se desarrollan robots de servicio de propósito general. Particularmente en 2011 comenzó el proyecto del robot de servicio Justina, con el que se realiza investigación en distintas áreas de la robótica.

En este laboratorio participan alumnos de distintos niveles académicos en distintas áreas de especialización, desde nivel licenciatura hasta doctorado y post-doctorado, con un equipo multidisciplinario que ha incluido en distintos períodos de tiempo estudiantes de diferentes ingenierías, e incluso estudiantes de algunas ciencias sociales y biológicas. El trabajo que se realiza aquí, frecuentemente consiste en desarrollos de tesis y servicio social, por lo que es frecuente que se incorporen nuevos estudiantes, a la vez que otros se vayan al concluir sus trabajos.

Por esta razón, uno de los objetivos de esta tesis es crear un planeador al que se le envíen comandos de alto nivel para realizar los planes y sea más fácil para los nuevos estudiantes construir los planes.

Objetivos

Los objetivos principales del presente trabajo son: el desarrollo de la planeación de acciones de un robot de servicio con un diseño que facilite la ejecución de los planes.

Particularmente, las características buscadas en el motor de planeación de acciones son las siguientes:

- Mediante el uso del ambiente gráfico de ROS se realizara la simulación de primitivas de movimiento esenciales para la ejecución de un plan.
- Construir y ejecutar planes a través de comandos de lenguaje natural, cuya sintaxis ofrece pocas especificaciones acerca de las acciones que el robot deberá realizar para completar el objetivo.
- Que los planes generados tengan la posibilidad de seguir adelante aunque alguna tarea halla fallado.
- Que el motor de planeación genere planes que manejen situaciones que otros planeadores no podrían.
- Facilitar la implementación y programación de nuevos planes para otro tipo de escenarios.

Organización de la tesis

En este primer capítulo se ha presentado una introducción a los problemas que pretende solucionar este trabajo, las motivaciones, y los objetivos planteados. En el capítulo 2, los antecedentes que darán pauta a este trabajo, así como otros trabajos relacionados y el estado del arte. El capítulo 3 presenta los detalles de la representación de los planes para el motor de planeación de acciones. En el capítulo 4 se explica la implementación del motor de planeación de acciones. Posteriormente, el capítulo 5 presenta los resultados obtenidos en las pruebas “general purpose service robot” y “open challenge” de la categoría @home de la RoboCup, comparando el desempeño del simulador contra el desempeño del robot de servicio real. El capítulo 6 presenta las conclusiones.

CAPÍTULO 2

ANTECEDENTES

En este capítulo de la tesis se hablará acerca del estado del arte en la planeación de acciones, la planeación clásica, la planeación con búsqueda heurística y la planeación probabilística; del alcance que se ha logrado; así como, de las dificultades y los problemas que aun quedan por resolver.

2.1 Estado del arte

En muchas de las investigaciones de Inteligencia Artificial que involucran planeación clásica se suele usar STRIPS y SAS+.

STRIPS (Stanford Research Institute Problem Solver)[1] fue desarrollado en la universidad de Stanford en 1971, es una herramienta que funciona como solucionador de problemas y se basa en la búsqueda de espacios de modelo del mundo con la finalidad de encontrar un modelo en el que se alcance el objetivo buscado. STRIPS asume que para cada modelo de mundo existe un conjunto de operaciones, cada una de las cuales transforma ese modelo para alcanzar a otro. La tarea de STRIPS es encontrar la composición de operadores que transformen un modelo de mundo inicial dado en otro hasta alcanzar un estado objetivo. El modelo de mundo incluye un gran numero de hechos y relaciones que tienen que ver con la posición del robot, la posición y atributos de objetos, espacio libre y limites; así que para representar el modelo de mundo se usan predicados de primer orden.

SAS+ [2] es una herramienta que se puede ver como una variación de STRIPS, la cual se creó con el propósito de reducir el enorme espacio de búsqueda en problemas de planeación. Con el uso de restricciones locales y globales así como la reducción de expresividad para representar el modelo del

mundo, se ganó eficiencia en la obtención de los operadores que permitirán alcanzar un objetivo especifico dado un estado inicial. Huang, Chen y Zhang en [3] afirman que SAS+ representa el modelo de mundo usando variables de estado multi valor en lugar de usar hechos proposicionales como en STRIPS. Además, SAS+ ha ayudado en investigaciones para encontrar heurística en [4] y [5], landmarks en [6], nuevos modelos de búsqueda en [7] y fuertes restricciones de exclusión mutua en [8].

Otra forma de mejorar la eficiencia computacional en la solución de problemas de planeación fue mediante el aprovechamiento de la heurística, que hace uso de conocimiento especial acerca del dominio del problema, siendo este representado por un grafo. En 1968 describieron por primera vez el algoritmo A∗ investigadores del Instituto de investigación de Stanford [9], el cual demostró tener un buen rendimiento ya que utiliza heurística para guiar las búsquedas. A∗ utiliza la búsqueda primero el mejor, cuyo objetivo es encontrar el camino de menor costo desde un nodo inicial hasta un nodo objetivo. Para esto, A∗ debe recorrer nodos del grafo que representa el problema, esta acción generará un árbol de caminos parciales. Las hojas de este árbol se alojan en una cola de prioridades que ordena los nodos hoja mediante una función de costo, que combina una estimación heurística del costo de alcanzar un objetivo y la distancia recorrida desde el nodo inicial. Se debe tener muy en cuenta que para encontrar el camino de menor costo, la función de heurística debe ser admisible, lo que significa que no se debe sobrestimar el costo para alcanzar el siguiente nodo objetivo mas cercano. La función de heurística es específica para cada problema en particular, por lo que se debe mejorar y adecuar para obtener un buen rendimiento del algoritmo.

En 1996 Culberson y Schaeffer introdujeron heurística basada en base de datos de patrones (PDB) [10], pero fue hasta 2001 que se usaron en problemas de planeación por Edelkamp [11]. En la actualidad lo mas utilizado son las “Incremental PDB” creadas por Haslum et al. en 2007 [12]. Sin embargo, a diferencia de la situación en otras áreas de aplicación de búsqueda heurística, las PDB’s se han quedado rezagadas en el estado del arte en planeación. Recientemente, la heurística merge-and-shrink [13] tiene el mejor

rendimiento en planeación heurística basada en abstracción (homomórfica).

Como la mayoría de los sistemas de planeación óptimos están basados en búsquedas en un espacio de estado con A∗ y una heurística admisible, se dio inicio a encontrar una forma de derivar la heurística automáticamente. Así es como surgió merge-and-shrink (M&S) [14], enfocado en abstraer y al mismo tiempo reducir el espacio de estados para producir una heurística consistente y admisible. El espacio de estados abstracto es construido de forma incremental, iniciando con un conjunto de abstracciones atómicas correspondientes a variables individuales, entonces, iterativamente, se mezclan(merging) dos abstracciones – reemplazándolas con su producto sincronizado – y se reducen (shrinking) – dos pares de estados se agregan en uno solo. Entonces, a pesar del tamaño exponencial del espacio de estados, M&S permite seleccionar pares individuales de estados para unirlos.

En 1994 se formalizó el uso de redes jerárquicas de tareas (HTN, Hierarchical Task Networks) [15], con el objetivo principal de reducir la brecha entre técnicas de planeación de IA, tales como la planeación con STRIPS, y técnicas de investigación de operaciones para la administración de proyectos y calendarización. Por lo que existen ciertas similitudes entre la planeación HTN y la planeación STRIPS. Como ya se había mencionado en STRIPS, el objetivo es encontrar una secuencia de acciones que lleven al mundo a un estado que satisfaga ciertas condiciones; el plan se lleva a cabo buscando operadores que tengan los efectos deseados. En contraste, la planeación HTN, busca planes que completen las tareas de red, tareas que no solo contemplan estados meta, y el plan se lleva a cabo a través de la descomposición de tareas y resolución de conflictos.

Posteriormente aparecieron las máquinas de estados jerárquicas para ejecutar los planes paso a paso sin haber generado el plan por completo. Los planes se construyen de forma jerárquica basándose en el enfoque HTN, y la ejecución utiliza un enfoque greedy (buscar el camino de menor costo). Por esta razón las máquinas de estados jerárquicas, son muy utilizadas porque permiten realizar acciones paso a paso, observar los efectos de estas acciones, y tomar decisiones dependiendo de los resultados.

SMACH es un módulo de ROS (Robot Operating System, http://www.ros.org/), cuya arquitectura permite la creación de conductas complejas para un robot mediante el uso de maquinas de estados jerárquicas [16]. SMACH es una librería basada en el lenguaje de programación Python, por lo que no depende de ROS y se puede utilizar en proyectos de Python independientes. Este modulo también tiene la capacidad de integrarse con la librería actionlib y cuenta con un visualizador para ver con introspectiva las maquinas de estados. La figura 2–1 muestra el ejemplo de una maquina de estado usada para coordinar acciones que permitan al robot PR2 cargar su propia batería en una conexión estándar de electricidad.

GOLEM es un grupo interdisciplinario perteneciente a la UNAM que también trabaja con robots de servicio, el cual hace uso de Redes de transición recursivas (Recursive Transition Networks, RTN), las cuales tienen un enfoque parecido al de las máquinas de estados jerárquicas. SitLog [17] es el nombre del framework que el grupo GOLEM desarrolló, el cual tiene la finalidad de proveer flexibilidad y abstracción a la descripción de tareas que realizará el robot y su implementación; además, modelan distintas situaciones en las que podría estar el robot y definen transiciones a otros estados o situaciones dependiendo del nivel de confianza esperado para manejar cierta situación con éxito.

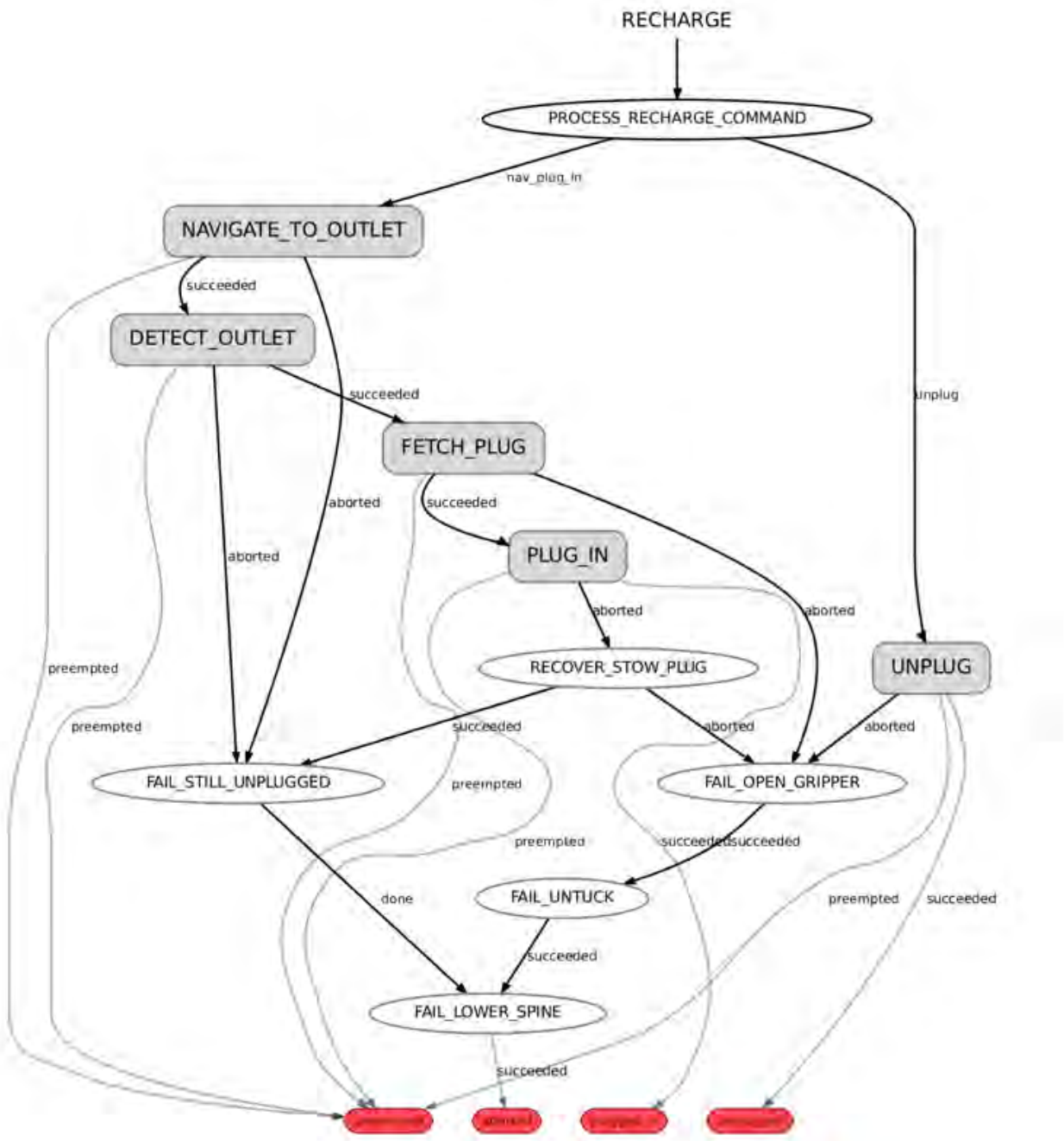


Figura 2-1. Máquina de estados que coordina las acciones del robot PR2 para recargar automáticamente su batería [18].

En 2012 investigadores de la Universidad de Feiburg presentaron PROST, un sistema de planeación probabilístico basado en el algoritmo UTC (Upper confidence bounds applied to tres, 2006) [19], el cual es utilizado en muchos problemas de acción bajo incertidumbre. La estructura de datos empleada por el algoritmo es un árbol de decisión, donde cada nodo representa una acción y los caminos hacia los nodos descendientes tienen una probabilidad asociada. Una consulta al algoritmo UTC siempre devolverá una decisión no aleatoria y se puede establecer un tiempo de ejecución para finalizar el algoritmo. Durante este tiempo el algoritmo realizará un despliegue de resultados de acciones basadas en su probabilidad. UTC se considera un algoritmo de tipo greedy, por lo que sus decisiones están basadas en los sucesores que lleven a grandes recompensas (mayor probabilidad de éxito). Sin embargo UTC tiene una limitación y es que se basa en búsquedas en profundidad, mientras converge a un objetivo óptimo en la práctica se necesita un número grande de iteraciones para que el algoritmo converja, es por eso que se establece un tiempo para la ejecución del algoritmo, pero hay que asegurarse que el tiempo dado es lo suficientemente apropiado para obtener resultados satisfactorios.

En lo que respecta al laboratorio de Bio-Robótica actualmente se utilizan máquinas de estado jerárquicas para definir los planes y ejecutar las acciones que el robot debe realizar. La arquitectura del robot esta basada en el modelo VirBot [20], propuesto por Savage et al. en 2007, utilizando ROS para mantener la comunicación entre los distintos módulos de la arquitectura. La figura 2-2 muestra el diagrama del modelo VirBot que se utiliza en el laboratorio. Este trabajo de tesis se enfoca en las áreas de planeación, ejecución y supervisión mostradas en la figura 2-2.

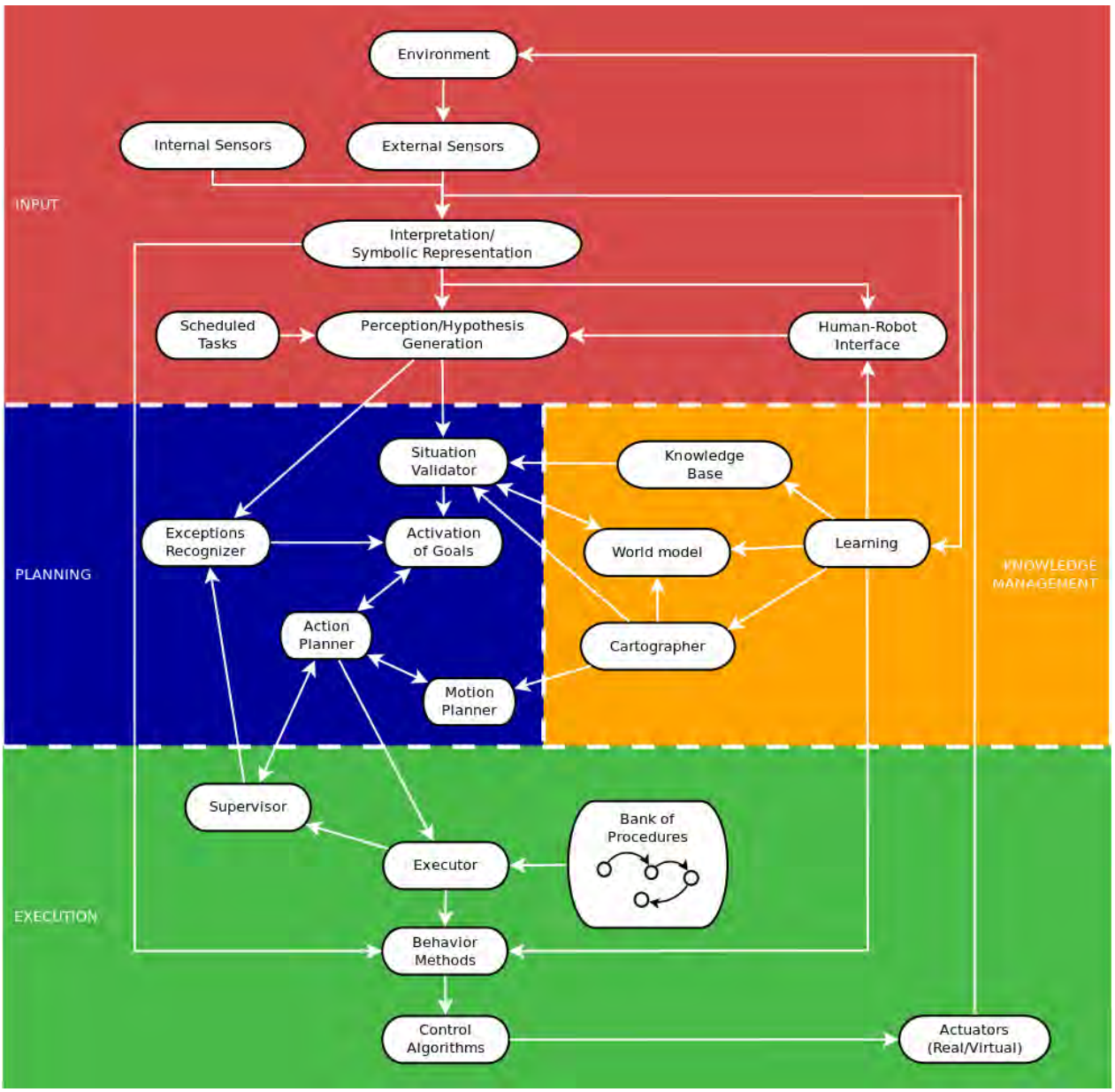


Figura 2-2. Diagrama del modelo VirBot.

ROS proporciona herramientas de paso de mensajes que se ocupan para comunicar distintos módulos de robot utilizando llamadas a procedimientos remotos a través de tópicos y servicios. De esta forma, el planeador es capaz de enviar comandos a otros módulos para realizar las tareas solicitadas.

2.2 Planeación clásica

Antes de realizar cualquier algoritmo para la planeación, es necesario tener un modelo que represente de manera apropiada el ambiente donde se desarrollan las tareas. El modelo no solo debe representar el entorno, si no que, debe tener un alto grado de precisión y abstracción para indicar las posibles acciones, sucesos o eventos que se puedan producir y poder manejar los cambios que resultaron de estas acciones.

El modelo que se utiliza para cubrir los requerimientos dichos anteriormente es un modelo dinámico, muy útil para representar conceptos básicos necesarios en el modelado del problema de planeación y así una vez terminado el modelo se procede a diseñar el algoritmo de planeación.

Respecto a la planeación clásica, el método más utilizado es SAS+, que es una versión mejorada de STRIPS. La planificación de tareas o también llamado problema de planeación en SAS+ tiene sus particularidades las cuales se definen a continuación.

Primero definiremos el **problema de planeación** como una tupla $\Pi = (V, O, s_0, S_*)$ donde:

V: es un conjunto finito de estados.
O: es un conjunto finito de operaciones (acciones) "o" con:
- Precondiciones $pre(o)$.
- Efectos $eff(o)$.
- Costos $\cos t(o)$.

Las operaciones se escriben como $(pre(o), eff(o), \cos(o))$.
S_0: El estado inicial.
S_*: La descripción del objetivo.

Una definición informal del problema de planeación seria:

Dado un problema de planeación
Encuentra un **plan** (secuencia de acciones) comenzando en el estado inicial hasta el estado objetivo (de otra forma muestra que el plan no existe).
Restricciones adicionales a tomar en cuenta:
Minimizar el costo del plan:
Minimizar la suma de costos.
Minimizar el número de acciones a realizar.

Ahora que conocemos la definición de un problema de planeación podemos darle una semántica formal completa a través del sistema de transiciones (también llamado espacio de estados). Dicho sistema es el modelo que se utiliza para representar el ambiente donde tendrá que interactuar el robot y sobre todo se utiliza en conjunto con el problema de planeación clásica descrito anteriormente.

Un **sistema de transiciones** se define como una tupla $\Theta = (S, L, T, s_0, S_*)$ donde:
S: es un conjunto finito de estados.
L: es un conjunto finito de etiquetas de transición,
Cada transición λ tiene asociado un costo $\cos t(o)$.
$T \subseteq S\ x\ L\ x\ S$ es una función de estado transición.
$s_0 \subseteq S$: estado inicial
$S_* \subseteq S$: estado objetivo

Un sistema de transiciones puede representarse también como un grafo dirigido donde los nodos representan los estados y los arcos las transiciones que estarán marcadas con las etiquetas de transición que habilitan la transición de un estado a otro. También hay que tomar en cuenta que para la función estado transición puede existir un estado $\in$ igual a un estado s, para ir de s a $\in$ se necesita de una acción a, además puede existir un estado e al cual se puede llegar desde un estado s sin aplicar una acción. Por lo que podemos reescribir la función de estado transición $T(s, a, \in)$ como $T(s, a)$ y $T(s, a, e)$ como $T(s, e)$.

Ahora bien si al sistema de transición descrito anteriormente le inducimos el problema de planeación se consigue la siguiente representación:

Un problema de planeación Π induce un sistema de transición $\Theta(\Pi)$ dando como resultado lo siguiente:

Estados: estados de Π.

Etiquetas de transición: operaciones de Π (con la misma función de costo).

Transiciones: transición (s, o, t) presente si:

$pre(0) \subseteq s$

$eff(o) \subseteq t$ y

Las variables en $pre(0) \subseteq s$ no coincidan con $eff(o) \subseteq t$.

Estado inicial: Estado inicial de Π.

Estado Objetivo: todos los estados s que coincidan con el objetivo de Π

De este modo representamos el problema de planeación a través del sistema de transiciones donde las acciones contribuyen a la evolución del plan a través de distintos estados.

Por último, se debe mencionar que el sistema podría no ser completamente observable, dicho de otra forma, el robot no tiene la información completa de las características del ambiente todo el tiempo, lo cual complica la planeación. Por este motivo hay que considerar métodos probabilísticos o técnicas que puedan realizar una estimación confiable que permita solventar estos problemas.

2.3 Planeación con abstracción heurística

Como se mencionó en el *estado del arte*, se buscó otra manera de resolver el problema de planeación de manera más rápida y con mejores resultados, por lo cuál se creo el algoritmo A*, que en conjunto con una heurística admisible ofrecen una Planeación óptima a través de una búsqueda heurística.

Una heurística admisible se define como:

Heurística: $h : S \longrightarrow \mathbb{R} \geq 0 \cup \{\infty\}$

h(S): Es la estimación del costo total desde un estado inicial hasta el objetivo.

S: es el mapeo de todos los estados visitados para encontrar la solución.

Admisible: no se debe sobrestimar el costo para alcanzar el nodo objetivo.

Normalmente la estimación de la heurística para obtener el costo total del mapeo de estados se obtiene como la **abstracción** de un espacio de estados real, por ejemplo, si se quiere hacer un viaje a través de distintos puntos en una ciudad, las distintas paradas serian los nodos del grafo y la heurística seria la distancia que existe para llegar a ese lugar en concreto o también se podría considerar como heurística el tiempo que tardas en llegar; de ambas formas la heurística nos estará representando un costo, el cuál nos servirá para planificar el recorrido por la ciudad de forma que el costo sea el mínimo posible.

Formalmente una **abstracción** en un sistema de transición Θ con estados S se define como la función:

Abstracción: $\propto : S \longrightarrow S'$

$\propto (s)$: **estado abstracto** para un estado **s** concreto.

La idea de la representación es que ofrece la distinción entre los estados s1 y s2 si ambos presentan el mismo **estado abstracto**, es decir cuando se tiene que $\propto(s1) = \propto(s2)$.

Finalmente, si inducimos el concepto de **abstracción** al sistema de transición, como se hizo con el problema de planeación clásica descrito en el capítulo anterior, obtenemos lo siguiente:

La abstracción $\propto$ para un sistema de transición concreto $\Theta = (S, L, T, s_0, S_*)$ induce el sistema de transición abstracto $\propto(\Theta)$, donde:

Estados: $\{\propto(s) \mid s \in S\}$.
Etiquetas de transición: L (las mismas que el original).
Transiciones: transición $\langle \propto(s),\ \ell \propto(t) \rangle$,
Inducida por cada transición original $\langle s,\ \ell,\ t \rangle \in T$.
Estado inicial: $\propto(s_0)$.
Estado objetivo: estado objetivo $\propto(S_*)$,
Inducido por cada estado objetivo $s_* \in S_*$.

Abstracción heurística: $h^{\propto}(s)$ = costo al objetivo desde $\propto(s)\ en\ \propto(\Theta)$.

De este modo tenemos un sistema de transiciones donde se ha añadido la abstracción de una heurística particular a su representación; esta representación ahora se puede usar en conjunto con el algoritmo A* para encontrar una solución optima al problema de planeación.

Para que una abstracción resulte útil, hay que tomar en cuenta los siguientes aspectos:

Puntos conflictivos para el uso de la abstracción en problemas de planeación:

- La obtención de la heurística informativa que minimice el costo y que sea adecuada al problema que se este resolviendo.
- Mantener una representación pequeña.

Una abstracción tiene una representación pequeña si cuenta con:

- Pocos estados abstractos.
- Una codificación breve, precisa y concisa para $\propto$.

La abstracción no solo se usa para la planeación con el algoritmo A*, recientemente se utilizan otros tipos de clases de abstracción para el estudio de planeación y búsqueda heurística, que se listan en orden de generalidad a continuación:

- Proyecciones (Bases de datos de patrones).
- Abstracciones de dominio.
- Abstracciones cartesianas .
- Abstracciones para el método Merge and shrink.

2.4 Redes jerárquicas de tareas

La planeación con Redes Jerárquicas de Tareas (Hierarchical Task Network, HTN) consiste en la elaboración de un plan compuesto por un número finito de tareas, dichas tareas deben cumplir con la restricción de tener una representación abstracta pequeña, posteriormente cada tarea es sustituida recursivamente por una red de tareas de menor jerarquía hasta que el plan queda descompuesto en una serie de acciones primitivas.

Las acciones primitivas corresponden a las acciones vistas en el problema de planeación clásica. Las tareas deben ser sometidas a diferentes métodos que describen y realizan su descomposición en otras tareas y/o acciones; además, una tarea puede ser sometida a diferentes métodos obteniendo distintas acciones a realizar dependiendo del estado actual del mundo.

A continuación se definirá la construcción de un plan con HTN.

Una red de tareas es un par w = (U,C), donde:

U: Es un conjunto de tareas.

C: Es un conjunto de restricciones.

Al realizar la planeación con HTN hay que tener en cuenta que las precondiciones y efectos de las acciones están generalizados como restricciones en los métodos. Cada restricción debe especificar un requerimiento que debe satisfacerse para avanzar en el plan y llegar a la solución del problema. Las restricciones utilizadas son las siguientes:

La restricción de precedencia se denota con la expresión $u \prec v$ donde:

u, v: Son tareas o acciones primitivas y quiere decir que todas las acciones en u deben ejecutarse antes de comenzar con la primera tarea de v.

La restricción before se denota como $before(U', l)$, donde:

$U' \supseteq U$: Es un conjunto de tareas.

l: Es una literal. La literal l debe ser cierta en el estado inmediato anterior a la primera tarea en U'.

La restricción after se denota como $after(U', l)$, donde:

$U' \supseteq U$: Es un conjunto de tareas.

l: Es una literal. La literal l debe ser cierta en el estado inmediato posterior a la ejecución de la última tarea en U'.

La restricción between se denota como $between(U', U'', l)$, donde:

$U', U'' \supseteq U$: Son un conjunto de tareas.

l: Es una literal. La literal l debe ser cierta en el estado inmediato posterior a la ejecución de la última tarea en U', debe ser cierta en el estado inmediato anterior a la primera tarea en U'', y en todos los estados intermedios.

Un método de HTN se define como una tupla $n = (nombre(m),\ tarea(m),\ subtareas(m),\ restricciones(m))$, donde:

$nombre(m)$: es una expresión de la forma $n(x_1, \dots, x_k)$ donde n es un símbolo de método único y $x_1, \dots, x_k$ son todos los símbolos de variables que ocurren en m.

$tarea(m)$: es una tarea o acción no primitiva.

$(subtareas(m),\ restricciones(m))$: Es una red de tareas.

La descomposición de una red de tareas se realiza de la siguiente manera: Sea w = (U, C) una red de tareas, $u \in U$ una tarea, m una instancia de un método y tarea(m) = u. Entonces u sufre una descomposición inducida por m dando como resultado la red de tareas w':

$\omega' = \delta(\omega, n, m) = ((U - \{u\}) \cup subtareas(m), C' \cup restricciones(m))$, donde:

C': Es una modificación de C con las siguientes características:

- Para cada restricción de precedencia que contenga a u, se debe reemplazar con restricciones de precedencia a cada tarea en la lista de $subtareas(m)$. Por ejemplo si la tarea u se divide en $subtareas(m) = \{u_1, u_2\}$, se reemplazaría la restricción $u \prec v$ por las restricciones $u_1 \prec v$ y $u_2 \prec v$.
- En cada restricción before, after y between para las que exista un conjunto de tareas U' que contenga a u, se reemplaza U' por $(U - \{u\}) \cup subtareas(m)$. Por ejemplo si $subtareas(m) = \{u_1, u_2\}$, entonces se reemplazaría la restricción $after(\{u, v\}, l)$ por $after(\{u1, u2, v\}, l)$.

Un dominio de planeación HTN se define como un par $D = (O, M)$, donde:

O: Es un conjunto de operadores.

M: Es un conjunto de métodos HTN.

Y un problema de planeación HTN se define como $P = (s_0, \omega, O, M)$, donde:

s_0: Es el estado inicial.

ω: es la red de tareas inicial.

Ahora si w = (U, C) es primitivo, es decir, contiene solo acciones primitivas, entonces un plan $\pi = < a_1, \ldots, a_k >$ es una solución para un problema P si existe una instancia (U', C') de (U, C) y un ordenamiento de las tareas en U' tal que todas las siguientes condiciones se satisfacen:

- Las acciones en π son las tareas $u_1, u_2, \ldots, u_k$.
- El plan π es ejecutable en el estado s_0.
- El ordenamiento satisface las restricciones de precedencia en C'.
- Por cada restricción $before(U', l)$ en C', l es cierta en el estado s_{i-1} que precede inmediatamente a la acción a_i, donde a_i es la primer acción de las tareas en U'.
- Por cada restricción $after(U', l)$ en C', l es cierta en el estado s_j producido por la acción a_j, donde a_j es la última acción de las tareas en U'.
- Por cada restricción $between(U', U'', l)$ en C', l es cierta en cada estado entre las acciones a_i y a_j, donde a_i es la última acción de las tareas en U', y a_j es la primer acción del las tareas en U''.

Si w = (U,C) no es primitiva, es decir, contiene al menos una tarea no primitiva, entonces π es una solución para un problema P si existe una secuencia de descomposición de tareas que pueda ser aplicada a w para producir una red de tareas primitiva w' tal que π es una solución para w'. Para más información acerca de las características y conceptos de HTN puede consultar Semántica para la planeación con HTN [21] y Planeación autónoma [22].

CAPÍTULO 3

REPRESENTACIÓN DE LOS PLANES

Los planes nos permiten hacer descripciones de las acciones que el robot esta destinado a realizar y su representación consiste en un lenguaje que proporcione el medio para especificar el comportamiento del robot, un mecanismo de interpretación para este lenguaje, y un mecanismo para manipular los planes que permita supervisar el curso de las acciones y determinar si el estado del mundo ha cambiado.

En este capítulo se describirá la representación de los planes de nuestro sistema. La representación de los planes consistirá de un Mecanismo de Comunicación y un Mecanismo de manipulación de los planes. El Mecanismo de comunicación, que se describe en la sección 3.1, define el lenguaje y la forma de interpretarlo. El Mecanismo de ejecución, que se describe en la sección 3.2, define la ejecución y supervisión del plan.

3.1 Mecanismo de comunicación

La habilidad de manejar conversaciones de lenguaje natural mejora significativamente las capacidades del robot, aumentando sus capacidades para percibir su entorno y realizar cambios en el. Por ejemplo, con ayuda de la comunicación con lenguaje natural el robot puede recibir un rango amplio de especificaciones de trabajo y adquirir información que no puede obtener utilizando sus sensores.

La comunicación se realizará por medio del generador de comandos aleatorios utilizado específicamente en la prueba general purpose service robot de la categoría @Home para Robots de servicio de propósito general. Los comandos son proporcionados al robot con lenguaje hablado, los cuales el robot reconocerá y convertirá en una cadena de texto; gracias a esta cadena de texto obtenemos una simplificación importante para la tarea de comunicación con

lenguaje natural entre Robot y usuario y la posterior interpretación del comando. El comando es una secuencia de palabras que conforman oraciones en Ingles escritas correctamente.

La interpretación del comando de voz se realiza en tres pasos:

3.1.1 Estructura gramatical de los comandos de voz.

Los comandos son proporcionados al robot a través de lenguaje hablado y, por medio del módulo de reconocimiento de voz, el comando es reconocido y guardado como una cadena de texto. El modulo de reconocimiento de voz cuenta con una gramática XML estandarizada para las pruebas de RoboCup@Home, la cual restringe los comandos que se pueden reconocer, ayudando de esta forma a aceptar comandos con una estructura definida; en este caso la estructura que deseamos es la de la prueba GPSR, que se puede consultar en el apéndice para obtener una descripción más detallada de la gramática XML. Entonces el robot puede recibir y realizar peticiones, resolver preguntas y dar información. Además los comandos de voz se obtienen al utilizar el generador de comandos aleatorios, el cuál es un software que conoce la gramática que se esta empleando y cuenta con una base de datos de vocabulario definido para crear oraciones, mediante combinaciones de palabras contenidas en dicho vocabulario, con una estructura definida. La definición de la gramática nos ofrece la ventaja de que es mas fácil interpretar la estructura de los comandos de voz y por consiguiente tener un mejor comportamiento al realizar la tarea especificada.

La figura 3-1 muestra un ejemplo de un comando de voz y la explicación de su estructura gramatical.

Comando de Voz: *"Find a person in the kitchen and answer a question "* De acuerdo a la gramática la oración se descompone como sigue:	
Acción 1	1 "verbo" **Find** 2 "objeto" **a person** 3 "lugar" **the kitchen**
Conector	4 "Acción 1" **and** "Acción 2"
Acción 2	5 "acción hablar" **answer a question**

Figura 3-1. Ejemplo de la estructura gramatical de un comando de voz. En la línea 1 se comienza a desglosar el contenido de la regla Acción_1 la cual contiene 3 elementos: la acción a realizar, sobre quien recae la acción y su localización (líneas 1 – 3).

Posteriormente en la línea 4 se hace una conexión con la palabra "and" hacia una nueva acción a realizar (definida por la regla Acción_2); en este caso la acción que se solicita es responder una pregunta(línea 5).

Este es solo un ejemplo de las oraciones que puede crear el generador de comandos aleatorios, para más ejemplos consulte la sección de comandos aleatorios en el apéndice. Las reglas para reconocer a cada comando pueden variar dependiendo el comando y todas ellas se encuentran en el archivo XML que define la gramática; además, la gramática es capaz de reconocer a todos los comandos ya que son generados a partir de la misma gramática que consta de un vocabulario con palabras validas definidas.

3.1.2 Análisis sintáctico del comando dado por el generador de comandos aleatorios

Después de que el comando es reconocido y guardado como una cadena de texto, su contenido es interpretado y transformado a una representación interna del robot. Con esta representación se facilita el razonamiento y la construcción del plan, además ofrece una conectividad lógica, jerárquicamente estructurada y capaz de distinguir entre la descripción del objeto en cuestión, la descripción de su localización, descripciones del estado del objeto y otro tipo de descripciones acerca del entorno y de las mismas acciones a realizar. Debemos recordar que el contenido del comando esta escrito en correcto Ingles que satisface las siguientes restricciones: solamente un conjunto de construcción de oraciones es aceptado por la gramática XML. El vocabulario se encuentra restringido a palabras escritas correctamente en el dominio de tareas domésticas. El módulo de Interpretación manda una señal de error si no es posible interpretar la sentencia. Estas señales de error pueden ser manejadas por estructuras de control que permiten al plan ser tolerante ante estos errores. El primer paso en la interpretación del comando de voz es el análisis sintáctico, para el cual se ha utilizado un software desarrollado en el laboratorio de Bio-Robótica en una tesis de maestría [23].

Un ejemplo del análisis sintáctico de un comando de voz se muestra en la figura 3-2.

Análisis Sintáctico
1. Primero el comando es separado en palabras clave aisladas llamadas tokens; estos tokens son insertados en secuencia a través de un autómata:
Keywords substitution: *"find a person in the kitchen and answer a question"*. **WORDS:** ['find', 'person', 'kitchen', 'and', 'answer', 'a_question']
2. El autómata nos da como resultado una cadena de tokens que representa la sintaxis del comando a realizar:
SINTAX: ['vrb', 'noun', 'noun', 'seq', 'vrb', 'noun']

Figura 3-2. Ejemplo del análisis sintáctico de un comando de voz.

El análisis sintáctico entonces nos proporciona una lista ordenada de tokens, la cual usaremos para inferir que secuencia de acciones se adapta mejor al comando solicitado.

3.1.3 Interpretación del contenido del comando

El siguiente paso en la interpretación del comando es identificar la secuencia de acciones más probable a llevar a cabo. Para este propósito se tienen varios templates destinados a ofrecer una serie de acciones dependiendo de la sintaxis obtenida.

En la figura 3-3 se muestra como ejemplo el template que más se adaptaría al comando de buscar a una persona e interactuar con ella.

Template Encuentra una persona y habla con ella	
{"params":	**["Action_find", "Find_person", "Find_location", "Action_talk", "Question"],**
"Action_find":	"Verbos específicos": [["find", "look_for"], "Sintaxis [verbo]": ["vrb"]],
"Find_person":	"Sintaxis [sustantivo]": [["noun"], "tipo de sustantivo: [persona]": ["person"]],
"Find_location":	"Sintaxis [sustantivo]": [["noun"], "tipo de sustantivo: [lugar]": ["place"]],
"Action_talk":	"Verbos específicos": [["speak", "answer", "tell", "say"], "Sintaxis [verbo]": ["vrb"]],
"Question":	"Sintaxis [sustantivo]": [["noun"], "tipo de sustantivo [pregunta]": ["question"]],

```
"conceptual_dependency":  “Acción 1”
                          "((action_type
                          find_person_in_room)
                          (params -Find_location-) (step 1))
                          " +"
                          “Acción 2”
                          ((action_type
                          wait_for_user_instruction)
}                         (params -Question-) (step 2))",
```

Figura 3-3. Template que corresponde a acciones para encontrar y hablar con una persona. Los elementos que comprenden al template son los siguientes:

Params: Es una lista que contiene elementos de una estructura sintáctica específica. Esta lista se compara con la que obtuvimos en el análisis sintáctico y dependiendo del nivel de similitud se asignara una calificación, que posteriormente se comparara con la de otros templates. El templete con la calificación mas alta es el que se elige para dar representación al plan.

Action_find: Es el primer elemento de la lista params e indica que en esa posición debe aparecer un verbo, el cual pueden ser “find” o “look_for”.

Find_Person: Es el segundo elemento de la lista params, indica que en esa posición debe aparecer un sustantivo de tipo persona.

Find_Location: Es el tercer elemento de la lista params, indica que en esa posición debe aparecer un sustantivo de tipo lugar.

Action_talk: Es el cuarto elemento de la lista params, indica que en esa posición debe aparecer un verbo el cual puede ser “answer”, “tell”, etc.

Conceptual_dependency: El framework de dependencia conceptual es un sistema lingüístico estratificado que intenta representar conocimiento, en este caso para comandos en lenguaje natural. El sistema se enfoca en conceptos más que en la propia sintaxis, y en entender en lugar de estructurar; además, se deben asumir inferencias ya que son fundamentales para un mejor entendimiento [24]. En este template el campo Conceptual_depency nos otorga la expresión final que representa el

entendimiento del comando de voz, o en otras palabras, nos devuelve una serie de tareas de alto nivel que el robot debe realizar para alcanzar su objetivo.

Implícitamente esta comparación entre la lista de sintaxis y los templates nos ayuda a definir los objetos, personas y localizaciones que el robot debe tomar en cuenta para llevar a cabo el plan. La figura 3-4 muestra el resultado final de la interpretación para un comando de voz dado.

Comando de Voz:

"Find a person in the kitchen and answer a question"

Expresión final:

```
(task (plan user_speech) (action_type find_person_in_room)
(params kitchen) (step 1))

(task (plan user_speech) (action_type wait_for_user_instruction)
(params a_question) (step 2))
```

Figura 3-4. Sentencia final para la interpretación de un comando de voz. La expresión anterior nos dice que el plan consta de 2 pasos: el primero, encontrar una persona en la cocina; el segundo, interactuar con la persona a través de contestar una pregunta.

3.2 Mecanismo de ejecución

La expresión final que arroja el mecanismo de comunicación nos proporciona un número de tareas a ejecutar, sin embargo la descripción de cada tarea es muy general y ambigua, por ejemplo tenemos la siguiente tarea:

> (task (**plan** user_speech) (**action_type find_person_in_room**) (**params kitchen**) (**step** 1))
>
> La tarea consiste en encontrar una persona en la cocina, pero esta tarea implica una serie de subtareas que no han sido especificadas. Tal es el caso de la navegación desde la posición actual del robot hasta la cocina, para posteriormente comenzar una rutina de reconocimiento de rostros hasta encontrar a una persona y finalmente acercarse a la persona.

Por lo tanto esta representación de los planes propone un plan de tareas de alto nivel, que se irán descomponiendo en tareas más específicas. A continuación se muestra un ejemplo de cómo se desglosarían las tareas de un plan:

- ➔ Encuentra una persona en la cocina (Tarea 1 find_person_in_room)
  - ➔ Navegación hasta la cocina
  - ➔ Encontrar una persona con reconocimiento de Rostros
  - ➔ Obtener localización de la persona
  - ➔ Acercarse a la persona
- ➔ Responde una pregunta (Tarea 2 wait_for_user_instruction)
  - ➔ El robot avisa que esta listo para recibir y responder una pregunta
  - ➔ El robot responde la pregunta

Se puede observar que cada tarea tiene una forma particular de dividirse en subtareas, por lo que cada tipo de tarea tiene una estructura única. Las tareas de alto nivel que se pueden generar son las siguientes:

1. **find_person_in_room**: El robot debe buscar una persona en un cuarto en específico.
2. **wait_for_user_instruction**: El robot espera por una instrucción de una persona como: responder una pregunta, presentarse, etc.
3. **update_object_location**: El robot debe navegar al cuarto donde se encuentra un objeto.
4. **get_object**: El robot realiza reconocimiento de objetos para encontrar uno en específico y lo toma con los actuadores.
5. **put_object_in_location**: El robot deja el objeto que lleva en sus actuadores en un lugar en específico
6. **handover_object**: El robot entrega el objeto que lleva en sus actuadores a una persona.

   Para una visión más detallada de estas tareas revisar el apéndice.

Se habrá de notar que con el número de tareas que se esta mostrando, los escenarios a considerar serian bastante limitados y por consiguiente tendrían que existir diferentes maneras de descomponer el plan, considerando cada situación que se pudiera presentar. Sin embargo, hay que mencionar que las tareas de alto nivel mencionadas anteriormente en general abarcan todas las acciones que una prueba de GPSR podría exigir en un momento determinado. Por otra parte, vamos a mostrar interés en las subtareas, que es en donde ciertas condiciones obligarán al robot a ejecutar o no ciertas acciones.

3.2.1 Representación del mundo

Las subtareas en realidad estarán condicionadas al estado en que se encuentra el mundo. Por ejemplo, algunas veces el robot tendrá que navegar a algún cuarto en el que ya se encuentra, otras veces tendrá que buscar a una persona en específico, o pedir información adicional acerca del objeto que necesita encontrar, etc.

Como estas condiciones, pueden existir otras más que influyen en la realización de más subtareas y también en la omisión de algunas, por esto es necesario tener conocimiento del estado del mundo. Por esto, se han creado templates con el propósito de definir los objetos y sus características particulares, y, mientras se vayan realizando las acciones del plan, poder modificar las características de los objetos que existen en nuestra representación del mundo.

A continuación se muestra el template que define a un objeto en nuestra representación del mundo:

```
(deftemplate item
   (type)
   (name)
   (status)
   (attributes)
   (pose (default 0 0 0))
   (grasp)
   (zone)
   (possession)
   (num)
)
```

El template nos ayuda a definir los objetos y asignarles sus correspondientes atributos. El campo **type** sirve para definir de que tipo de objeto se trata ya sea una persona, un mueble, un cuarto, etc. Además, el template nos da la facilidad de no tener que llenar todos los campos para cada objeto, ya que se puede dar el caso de que un objeto no tenga cierto atributo, en esos casos el campo adquiere por omisión un valor nulo.

Para obtener más detalles acerca del template que define los objetos del mundo consultar el apéndice.

CAPÍTULO 4

IMPLEMENTACIÓN DEL PLANEADOR

Como se mencionó en el capítulo anterior, el objetivo de este planeador es construir y ejecutar planes a través de comandos de lenguaje natural, cuya sintaxis ofrece pocas especificaciones acerca de las acciones que el robot deberá realizar para completar el objetivo; es decir, el planeador debe ser capaz de entender el comando y activar la ejecución de acciones de una manera menos procedural.

El planeador se comunicará con módulos de ROS que permiten el envío y recepción de mensajes, así como el uso de variables compartidas entre diferentes módulos de operación del robot.

En la implementación del planeador de acciones se utilizará el lenguaje de programación CLIPS cuyo paradigma se basa en un conjunto de hechos y reglas para la construcción de sistemas expertos. Los hechos activan las reglas que definen el flujo de ejecución, además de que a cada regla se le puede asignar un nivel de prioridad para que en caso de que se active otra regla al mismo tiempo la de mayor prioridad se ejecute primero.

El flujo de ejecución de los planes para esta implementación se logra utilizando hechos estructurados de CLIPS, es decir, el uso de templates que representan al mundo con campos que contienen información del objeto al que hacen referencia. Así como nuevos hechos estructurados que contengan tanto información del objetivo al que se hace referencia como su estado en el contexto de ejecución de cada plan. Con estos hechos podemos definir la máquina de estados principal del planeador y el template que los define se muestra en la figura 4–1 .

Hechos que definen la máquina de estados Principal (tamplate)	
(field cd)	Identificador o nombre de la tarea a realizar.
(field actor)	Agente que realiza la acción
(field obj)	Objetivo final que se desea alcanzar en este estado
(field from)	En que lugar inicia la acción
(field to)	En que lugar termina la acción
(field state-number)	Se asigna un número correspondiente al estado en que se realizara esta tarea.
(field name-scheduled)	Nombre del plan al que pertenece el estado

Figura 4-1. Template de los hechos que definen la máquina de estados principal. El campo **cd** nos ayuda a definir la tarea, el campo **actor** define quien realiza la acción, y el campo **state-number** define el estado en el que se debe realizar dicha tarea.

Los hechos aquí presentados sirven para detectar en que estado de ejecución se encuentra un plan.

4.1 Reglas que definen el ciclo de ejecución

La descomposición de los planes en tareas de alto nivel así como su ejecución se especifican a través de reglas con condiciones iniciales y resultados particulares. Sintácticamente se representan como reglas de CLIPS, y las podemos agrupar en reglas de ejecución, reglas de descomposición de tareas, reglas de finalización de tareas y reglas de cancelación.

Las **reglas de ejecución** deben tener precondiciones que restrinjan su activación, hasta que el estado del mundo cambie y cree nuevos hechos que permitan su activación. Los resultados obtenidos por estas reglas deben generar nuevos hechos especificando un orden jerárquico entre ellos. Estos hechos reflejan el estado del mundo después de la ejecución de una tarea o detectan algún evento.

Las reglas de ejecución deben incluir como condiciones iniciales:

(1) El hecho **(cd-task (state-number ?number))** que corresponda con la tarea a realizar en cierto estado.
(2) El hecho **(state (number ?number) (status ?active)),** donde ?number indica el estado actual que se encuentra activo y se vincula con la tarea a realizar.
(3) La restricción de que no exista un hecho **(state (status accomplished))** ya que no se trata de una regla de finalización.

En conclusión, las reglas de ejecución corresponden a tareas que están activas.

Las **reglas de descomposición** existen para desglosar una tarea de alto nivel en tareas más primitivas. La figura 4-2 muestra una plantilla de este tipo de reglas que generan subtareas.

Descomposición de la tarea de alto nivel: put_object_in_location	
Nombre de la regla	**plan_put_obj_in_loc**
Condiciones de Inicio	"Tarea de alto nivel": ?put_obj_loc "Objeto": ?obj "Lugar donde se depositara ?obj": ?place "número de la tarea de alto nivel": ?step
Hechos generados por la regla de descomposición	"primer subtarea: navegar hacia ?place" (plan **(number 1)**(actions **go_to_place** ?place))) "segunda subtarea: alinearse con ?place" (plan **(number 2)**(actions **attend** ?place)) "tercera subtarea: mover el brazo que sostiene a ?obj" (plan **(number 3)**(actions **move** manipulator ?obj)) "cuarta tarea: soltar ?obj" (plan **(number 4)**(actions **drop** object ?obj)) "hecho que indica de cuantas subtareas consta la tarea de alto nivel" (finish-planner ?name 4)

Figura 4-2. Ejemplo de una regla que descompone una tarea de alto nivel en subtareas. En este ejemplo la regla **plan_put_obj_in_loc** se divide en las subtareas: **go_to_place** que implica la navegación hasta la localización donde se debe depositar el objeto, la subtarea **attend** consiste en alinearse, de la mejor manera, con el lugar donde se

depositara el objeto por ejemplo alinearse con una mesa, **move** es una subtarea para la manipulación de los brazos del robot, con ella se indica al robot que mueva su brazo en una posición adecuada para soltar el objeto, finalmente la subtarea **drop** consiste en liberar el objeto sobre el lugar deseado.

Usualmente las tareas y subtareas son acciones primitivas que consisten en un conjunto de reglas que realizan llamadas a procedimientos remotos (Remote Procedure Call, RPC), siendo la principal función el que en algún momento deben producir un hecho que indique si la tarea fue exitosa o no lo fue.

En la implementación de esta tesis, los RPC corresponden con servicios y tópicos que se envían a los módulos del Robot utilizando ROS. Con la herramienta que aparece en el apéndice C se crea una conexión entre ROS y CLIPS, pero particularmente con la regla send-command de la herramienta BBCLIPS. La conexión esta compuesta básicamente por tres reglas: una regla inicial que envía el comando, una regla que recibe una respuesta fallida y una regla que recibe una respuesta exitosa.

Estas reglas de conexión hacen uso de hechos que dan información sobre el estado de ejecución de la tarea o subtarea. Existen tres hechos que dan información de este tipo: los hechos **waiting**, que se crean cuando se envía un comando y se espera por una respuesta; los hechos **BB_timer**, que se crean después de un lapso de tiempo, el cual se específica en la regla con la función setTimer; y los hechos **BB_answer**, que contienen la respuesta a un comando.

El hecho **waiting** se crea cada vez que se activa la función send-command para la activación de un procedimiento remoto y es destruido cuando se recibe una respuesta. La figura 4-3 muestra la definición del hecho waiting.

Definición del hecho Waiting (template)	
(slot cmd)	Este campo guarda el nombre de la llamada a procedimiento remoto.
(slot id)	Este campo contiene el identificador de la llamada a procedimiento remoto.
(slot args)	Guarda los argumentos que se utilizaron en la llamada a procedimiento remoto.
(slot timeout)	Almacena el valor del umbral de tiempo que se debe esperar para recibir respuesta de la llamada a procedimiento remoto.
(slot attempts)	Almacena el número de veces que ha transcurrido el umbral de tiempo establecido para recibir respuesta.

Figura 4–3. Definición del hecho waiting.

La función **setTimer** actúa como un temporizador, recibe como parámetro un valor de tiempo y este tiempo va en aumento hasta alcanzar un determinado valor, entonces se crea un hecho con la forma **BB_timer <symbol>**, donde <symbol> representa el timer (tiempo transcurrido) del que se trate.

La respuesta a un comando es enviada de ROS a CLIPS a través de un hecho con la siguiente estructura:

(**BB_answer** <command> <symbol> <answer> <params>)

Donde **<command>** representa el nombre del comando al que se da respuesta; **<symbol>** representa una instancia particular para hacer referencia a este comando, es útil cuando más de una tarea envíe el mismo comando; **<answer>** toma el valor 1 ó 0, indicando si la ejecución del comando fue exitosa o no respectivamente; y **<params>** es una cadena de tamaño variable cuyo contenido puede estar formado por parámetros tanto de salida como de respuesta. Para más información sobre este sistema de mensajes entre ROS y CLIPS consulte el apéndice C.

Las figuras 4-4, 4-5 y 4-6 muestran las reglas de llamada a procedimiento, respuesta positiva y respuesta negativa respectivamente para la tarea de confirmación de realización de un comando.

Llamada a procedimiento Remoto: "CONFIRMACION"	
Nombre de la regla	**exe_confirmation**
Condiciones de inicio	"Identificador de la tarea": (cd-task (cd confirmation))(name-scheduled ?name)(state-number ¿num))) "Nombre del comando a confirmar": (cmd_conf ?command))
Hechos generados por la regla	"se concatenan los argumentos a enviar, en este caso el nombre del comando a confirmar ?command": (bind ?argumentos (str-cat "" ?command)) "Con este hecho se realiza la llamada a procedimiento remoto": (send-ros ACT-PLN **cmd_conf** ?argumentos 6000 4)

Figura 4-4. Llamada a procedimiento remoto de la tarea CONFIRMACIÖN.

Respuesta positiva de “CONFIRMACION”	
Nombre de la regla	**conf_command**
Condiciones de inicio	“El hecho received debe contener el id de la llamada [cmd_conf] seguido de los argumentos y el último parámetro debe valer 1 para indicar que se trata de una respuesta positiva”: (**received** ?sender command **cmd_conf** ?plan ?steps **1**)
Hechos generados por la regla	“Se crea la nueva tarea a realizar” (cd-task (cd get_task) (state-number 3))) “se obtiene el número de pasos de los que consta el plan a ejecutar ” (num_steps (+ ?steps 1)) “se obtiene el nombre del plan que se intentara ejecutar” (plan_name ?plan)

Figura 4-5. Respuesta positiva de la llamada a procedimiento remoto de la tarea CONFIRMACIÖN.

Respuesta Negativa de “CONFIRMACION”	
Nombre de la regla	**conf_command**
Condiciones de inicio	“El hecho received debe contener el id de la llamada [cmd_conf] seguido de los argumentos y el último parámetro debe valer 0 para indicar que se trata de una respuesta negativa”: (**received** ?sender command **cmd_conf** ?arg **0**)
Hechos generados por la regla	“Se crea la nueva tarea a realizar” (cd-task (cd cmdSpeech) (state-number 1))) “Se genera el hecho que indica que no esta activa la creación de un nuevo plan” (plan_active no)

Figura 4-6. Respuesta negativa de la llamada a procedimiento remoto de la tarea CONFIRMACIÖN.

Una tarea puede hacer más de un llamado a un procedimiento remoto, ya que recibir una respuesta fallida no necesariamente significa el fallo de la tarea. Podría hacer falta realizar más intentos antes de que la tarea se considere como fallida, o realizar otras acciones que modifique el estado del mundo y generar respuestas que permitan la realización exitosa de la tarea. EL flujo de ejecución de planes esta diseñado de tal forma que se asegura que la ejecución (exitosa o fallida) de un acción termina y se crea el hecho que indica el estado final de la ejecución.

Para prevenir el caso de que el flujo de ejecución se detenga al estar esperando por una respuesta o un timer, existen unas reglas dentro del ciclo de ejecución con prioridad negativa, que indicaran que la tarea ha fallado,

aunque las reglas de la tarea no lo indiquen.

En este caso, antes de marcar el plan como fallido, se genera una notificación por medio del sintetizador de voz con la cual el robot anuncia al usuario que no puede finalizar la tarea.

La tarea **spg_say** se utiliza para activar y decir alguna frase a través del sintetizador de voz, pues recordaremos que el planeador esta diseñado para un robot de servicio el cual debe interactuar de manera natural y anunciar cuando no puede realizar una tarea.

Tanto para la ejecución de tareas como para su descomposición en subtareas, es necesario utilizar hechos como banderas que habiliten una u otra regla. Sin embargo, tras la ejecución de estas reglas, las banderas se deben borrar. Además es probable que se requiera crear hechos para actualizar la base de conocimientos y así mismo el estado del mundo, dependiendo siempre del resultado final obtenido de la tarea (exitoso o no exitoso).

Utilizando una notación similar a la utilizada en la planeación clásica, donde una acción corresponde a una tupla que contiene un conjunto de condiciones iniciales y un conjunto de resultados, podemos modelar una tarea como una tupla con los mismos elementos: con un conjunto de elementos que se producen si la tarea fue exitosa y un conjunto diferente de elementos que se producen si la tarea ha fallado.

Las **reglas de finalización** tienen como objetivo realizar acciones que permitan que el flujo de ejecución continúe en operación una vez que la tarea finalizó, ya sea con un resultado exitoso o fallido. Estas reglas no pueden generar otras tareas o subtareas, sólo son procesos que actualizan la base de conocimientos. Las reglas de finalización necesitan precondiciones similares a las reglas anteriores, debiendo existir un hecho que diga que la tarea esta activa pero acompañado de otro hecho (condicional) que indique que la tarea ha finalizado, entonces se genera un hecho que dice que la tarea ha sido completada y otro hecho que active la siguiente tarea a ejecutar.

Las reglas de finalización deben incluir como condiciones iniciales lo

siguiente:

(1) El hecho **(condition (conditional if) (arguments ?object status ?status) (state-number ?st)** que indica la condición de finalización de la tarea, dependiendo del estado ?status de ?object.

(2) El hecho **(state (number ?st)(status active))** que indica que la tarea aun esta activa.

A continuación se presenta una regla de finalización donde se actualiza la base de conocimientos para activar la nueva tarea a realizar:

Regla de finalización para una tarea exitosa	
Nombre de la regla	**exe_scheduled-if-conditional-true-status**
Condiciones de inicio	"Condición de finalización": (condition (conditional if) (arguments ?object status ?status_object) (state-number ?st)) "tarea actual activa": ?f1 <- (state (name ?name) (number ?st)(status active)) "tarea siguiente": ?f2 <- (state (name ?name) (number ?ts))
Hechos generados por la regla	"Se vuelve inactiva la tarea actual " (modify ?f1 (status inactive)) "Se activa la siguiente tarea a ejecutar" (modify ?f2 (status active))

Figura 4-7. Regla de finalización cuando la tarea actual fue exitosa.

Regla de finalización para una tarea que ha fallado	
Nombre de la regla	**exe_scheduled-if-conditional-false-status**
Condiciones de inicio	"Condición de finalización": (condition (conditional if) (arguments ?object status ?status_object) (state-number ?st)) "tarea actual activa": ?f1 <- (state (name ?name) (number ?st)(status active)) "tarea siguiente": ?f2 <- (state (name ?name) (number ?fs))
Hechos generados por la regla	"Se vuelve inactiva la tarea actual " (modify ?f1 (status inactive)) "Se activa la siguiente tarea a ejecutar" (modify ?f2 (status active))

Figura 4-8. Regla de finalización para cuando la realización de una tarea a fallado.

4.2 El algoritmo del planeador

El algoritmo de planeación consiste en un ciclo iterativo (ciclo de ejecución) que en cada estado revisa cuál será la siguiente tarea que se debe realizar. Cada iteración consta de cinco partes: recepción de un comando, interpretación, confirmación del comando, creación del plan y ejecución del plan completo. La figura 4–9 muestra el diagrama de flujo correspondiente al algoritmo general de planeación/ejecución.

La recepción de comandos consiste en realizar una llamada a un procedimiento remoto, donde el robot, por medio del sintetizador de voz, indica que esta listo para recibir una petición del usuario. El comando es pasado a una cadena de texto y analizado por el modulo de interpretación para conocer si se trata de un comando reconocible; al final se manda una respuesta de éxito o fallo a CLIPS con el propósito de avanzar a la siguiente tarea en caso de éxito o volver a pedir el comando en caso de fallo. La recepción de comandos de voz y la interpretación se describieron anteriormente con mayor detalle en el tema 3.1 “Mecanismo de comunicación”.

La siguiente tarea consiste en que el robot anuncie al usuario el comando que va a realizar y antes de comenzar necesita la autorización del usuario para proceder a realizar dicho comando. En caso de querer proceder con el plan, el usuario dará su confirmación al robot mediante las palabras “**robot yes please**”. Existen situaciones en las que el reconocedor de voz que posee el robot confundirá palabras y hará un reconocimiento erróneo del comando a ejecutar, entonces el usuario debe realizar una confirmación negativa con las palabras “**robot no please**”, de esta manera el robot no procederá a ejecutar el plan, y estará listo y en espera de recibir un nuevo comando de voz.

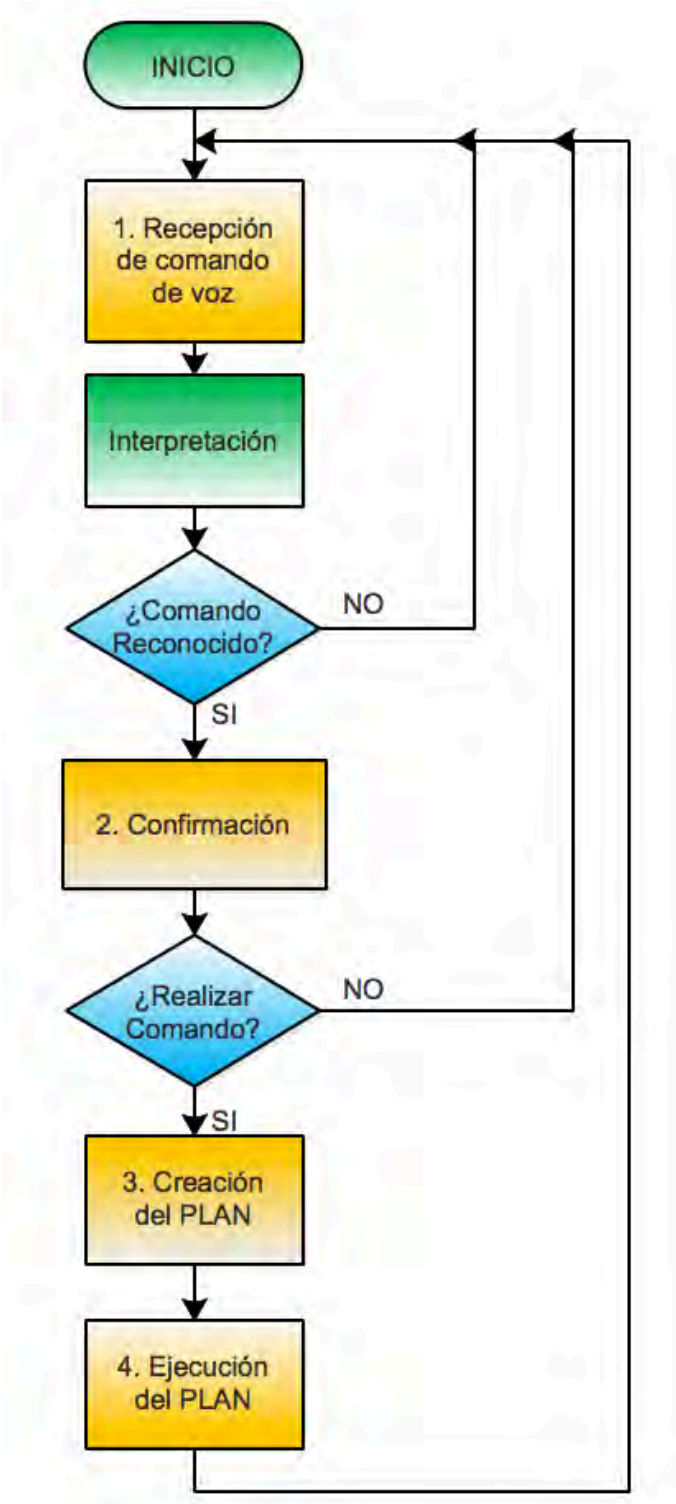


Figura 4–9 : Algoritmo general de planeación y ejecución.

Por simplicidad el algoritmo se presenta como un diagrama de flujo, pero cabe recordar que su implementación está hecha en reglas de CLIPS. La secuencia de ejecución de las reglas de CLIPS depende de las condiciones que activan a cada regla, por lo que es necesario implementar un mecanismo de control que permita ejecutar algoritmos a través de aserción y modificación de hechos en la base de conocimientos, de tal forma que se habiliten las siguientes reglas a ejecutar y deshabiliten las que no les corresponde ejecutarse aún y de esa forma mantener el ciclo de ejecución siempre activo.

En el resto del capítulo se describirá la parte de Creación del Plan y la parte de Ejecución del PLAN del Algoritmo general de planeación.

4.2.1 Creación de los planes

Una vez que se ha confirmado la realización del comando por parte del usuario, se habilitaran las reglas correspondientes para la creación del plan. La creación del plan contempla la activación de reglas para la obtención de tareas ordenadas, almacenadas en una pila, así como la activación de las reglas que descomponen las tareas en subtareas.

La obtención de las tareas concederán al planeador información necesaria para la solución del problema, información que se puede traducir en hechos y que se encuentra embebida en el diseño de reglas. En la figura 4–10 se muestra el diagrama de flujo con el algoritmo general para la creación de planes.

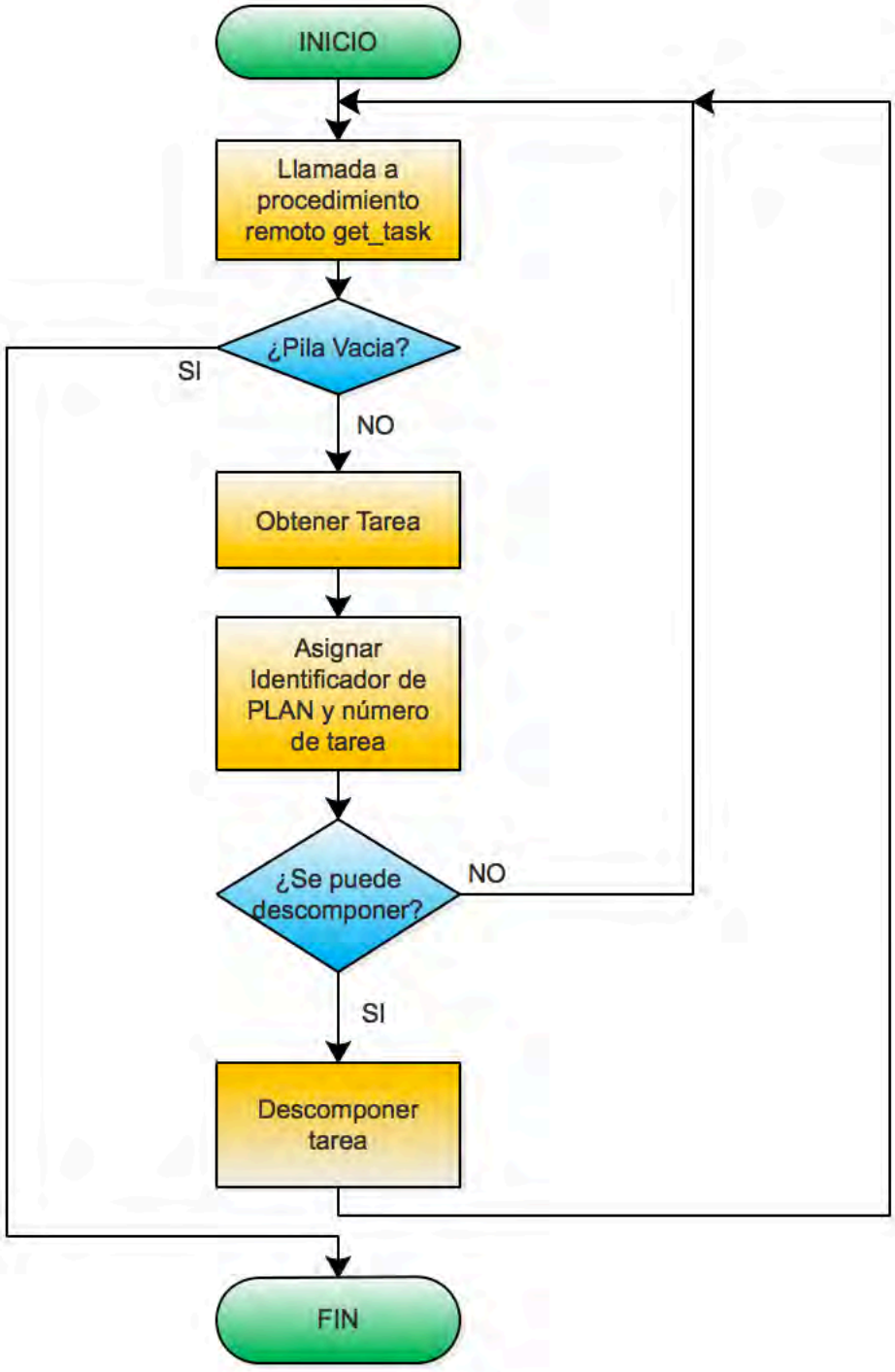


Figura 4-10 Diagrama de flujo para la creación de planes.

Es importante contemplar como se comporta esta parte del algoritmo y entender la relación que tiene con los otros procedimientos, asimismo se adquiere una

visión más amplia del enfoque pretendido en el diseño del planeador.

La principal función del algoritmo de creación de planes es resolver las dependencias necesarias para la ejecución de la tarea. Note que al descomponer una tarea de mayor jerarquía en sus correspondientes subtareas, estas se pueden asociar mediante un identificador de plan y el nombre de la tarea padre. La ejecución de las subtareas determinará el éxito de la tarea padre.

La descomposición de tareas en subtareas es un proceso que como resultado final nos regresa un subplan para lograr realizar una tarea en particular. De acuerdo a la situación actual del ambiente, el subplan podría descomponerse de una manera u otra.

Al final de este proceso tendremos creado el plan antes de que el robot ejecute cualquier acción para alcanzar su objetivo.

4.2.2 Ejecución del plan

El proceso de ejecución del plan consiste en una serie de llamadas a procedimientos remotos para realizar acciones tales como: navegar, ejecutar reconocimiento de rostros, ejecutar reconocimiento de objetos, activar el sintetizador de voz, mover el brazo, activar reconocimiento de voz, etc.

Después de realizar una llamada a procedimiento remoto, el proceso se queda en espera de recibir una respuesta positiva o negativa. Después de cierto tiempo de no recibir respuesta, el robot intentará ejecutar la siguiente tarea o subtarea, así mismo cuando después de tres intentos el robot obtenga una respuesta negativa intentara ejecutar la siguiente tarea o subtarea, esto es porque cabe la posibilidad de que la tarea que falló no afecte a la culminación de la siguiente tarea; por el contrario si la tarea fallida afecta en la culminación de la siguiente tarea los hechos en la base de conocimientos detectaran que no es posible culminar esa tarea, entonces se procede a intentar ejecutar la siguiente tarea del plan. La figura 4–11 muestra el diagrama de flujo del algoritmo de la ejecución del plan.

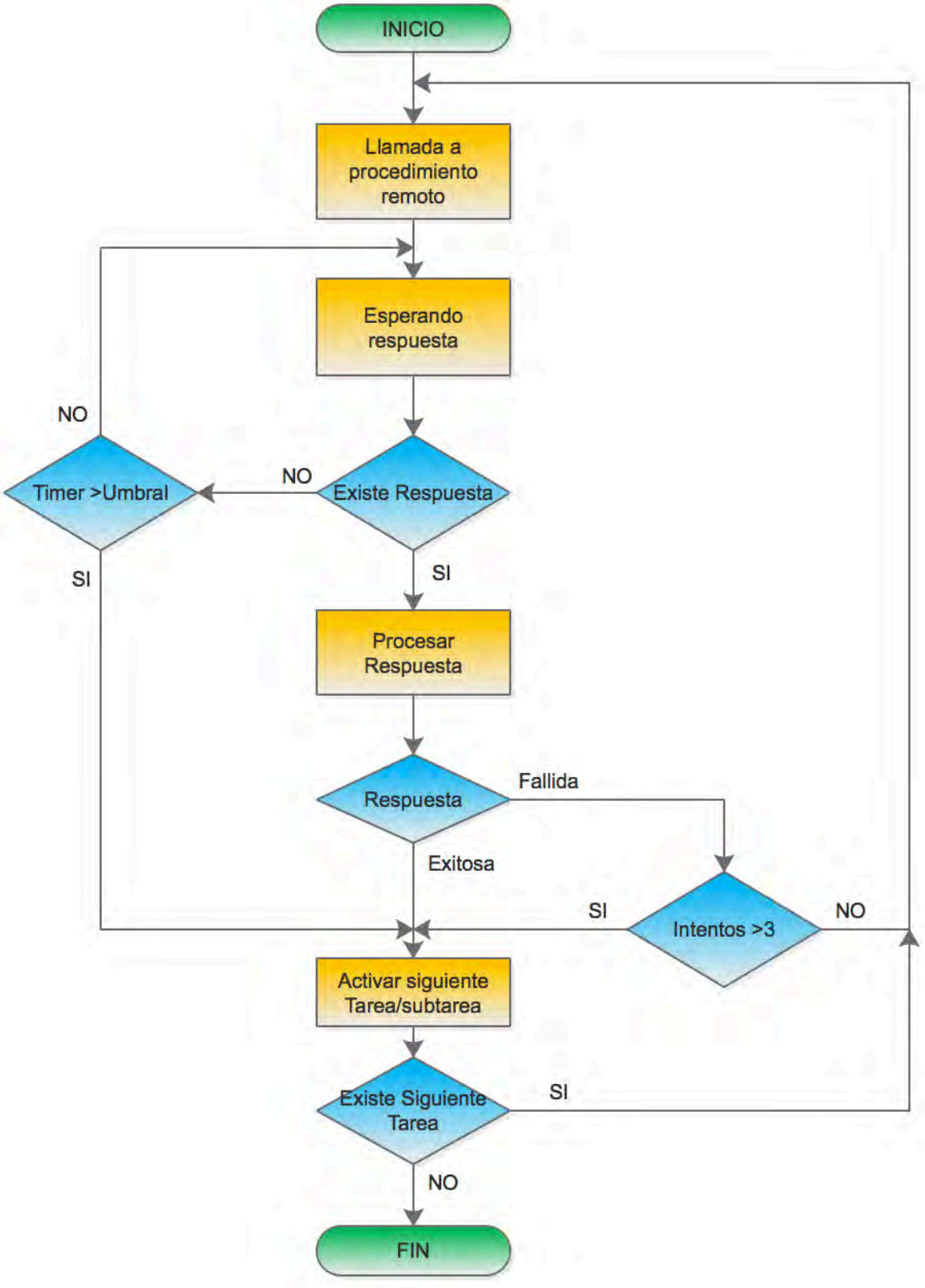


Figura 4–11: Diagrama de flujo del algoritmo de ejecución del plan.

Una tarea podría activarse varias veces debido a que después de la ejecución fallida de alguna subtarea se necesiten resolver algunas dependencias hasta obtener una respuesta exitosa.

La función de las subtareas es afectar directamente el ambiente real, a través de los sensores y actuadores del robot, y para acceder a ellos es necesario hacer uso de algunas interfaces mediante ROS. Es aquí donde se utilizan las llamadas a procedimientos remotos.

Otra forma en que puede terminar la ejecución de una tarea es si las tareas son canceladas por el motor de planeación. Que es el caso de exceder un limite de tiempo en la espera de una respuesta. De esta manera el robot pasa a estar en un estado disponible para que se lleve a cabo la siguiente tarea y permitir continuar con el ciclo de ejecución.

Entonces, estas reglas de finalización pueden tomar en cuenta el estado final de la tarea (exitoso o fallido) para realizar distintas acciones, o ignorarlo y realizar acciones independientes del resultado de la tarea.

Capítulo 5

PRUEBAS Y RESULTADOS

Cuantificar el desempeño del planeador de acciones desarrollado en está tesis es complicado, ya que se trata de procesos simbólicos con un diseño que en su mayoría presenta un componente subjetivo por la naturaleza y la falta de información a priori en algunos casos de las tareas que se deben realizar. El planeador de acciones fue diseñado para resolver problemas que abarcan un conjunto de escenarios, en específico escenarios de la prueba GPSR, para la resolución de otro tipo de problemas seria necesario hacer pequeñas modificaciones en la estructura del algoritmo general de planeación y ejecución.

Cabe mencionar que la eficiencia computacional no es el principal objetivo en esta tesis, sino que los planes que se generan tengan la posibilidad de seguir adelante aunque alguna tarea halla fallado, que los planes generados manejen situaciones que otros planeadores no podrían, y facilitar la implementación y programación de nuevos planes para otro tipo de escenarios.

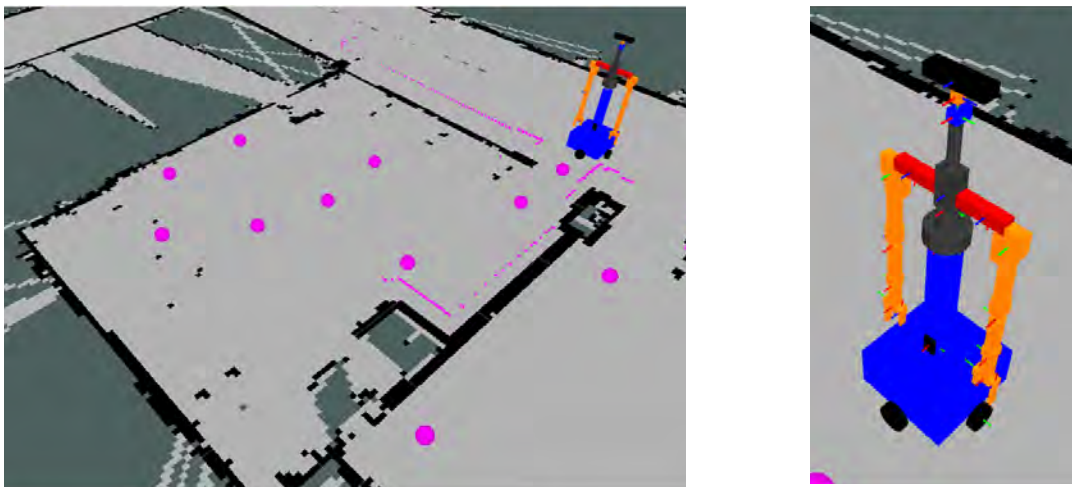
Los resultados de esta tesis se reportaran de la siguiente manera: se presentaran escenarios con condiciones particulares y se verificará si el plan generado puede ser manejado por el planeador de acciones descrito. Posteriormente se mencionaran algunas observaciones sobre el planeador.

5.1 Pruebas

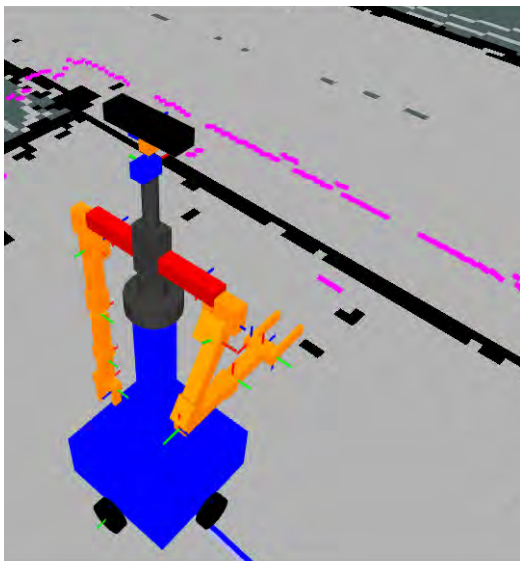
5.1.1 Simulación

El laboratorio de Bio-Robótica cuenta con un simulador gráfico el cual incluye la simulación de los movimientos más básicos del robot, tales como: la navegación, el movimiento de brazos, el movimiento de la cabeza, el reconocimiento de voz, y otros elementos parcialmente simulados, tales como: reconocimiento de objetos y reconocimiento de personas.

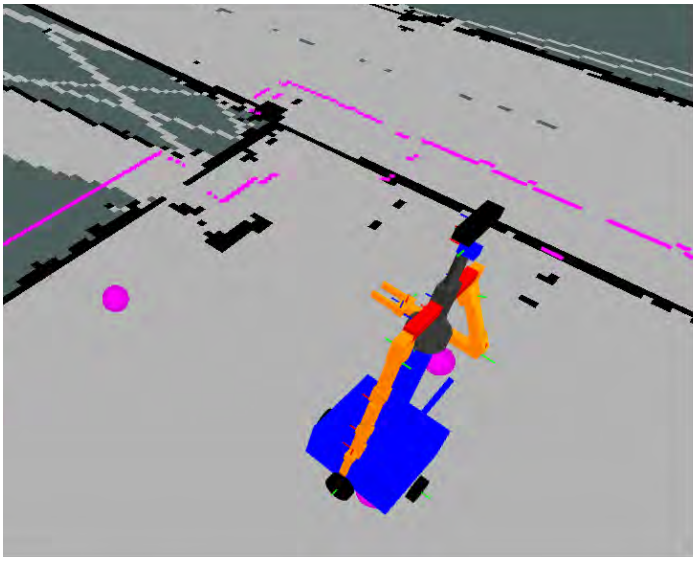
Gracias a las características que proporciona el simulador, la probabilidad de que el plan se lleve a cabo en la realidad, justo como sucedió en la simulación, es muy alta; para más detalles consulte el capitulo 5.2 Evaluación del rendimiento del planeador de acciones. La ventaja del simulador es que ofrece resultados parecidos a los reales y permite a varias personas realizar pruebas a la vez sin necesidad de utilizar el robot. Por estas razones las pruebas del GPSR y Open Challenge se ejecutaban primero en el simulador, con lo cuál se evidenciaban los errores en la planeación y se podían corregir casi inmediatamente. En la figura 5-1 se observa una muestra parcial de la apariencia del simulador y como el robot interactúa en el entorno gráfico.



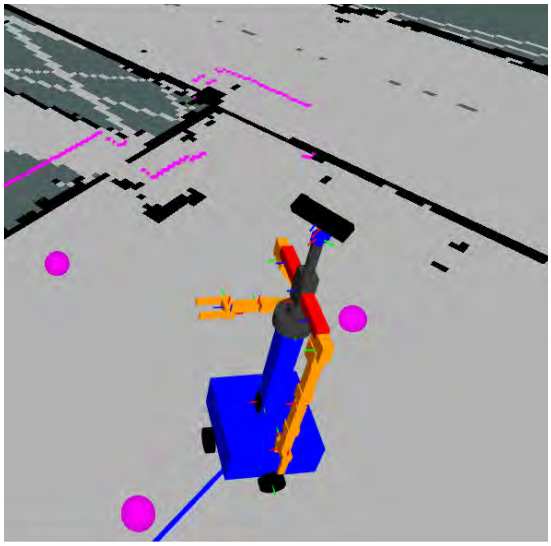
a) b)



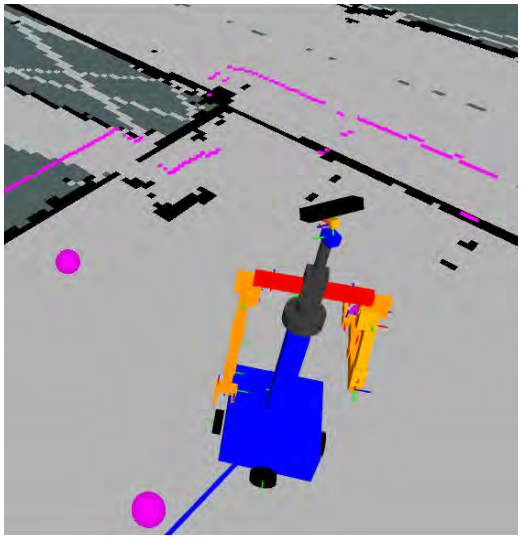
c)



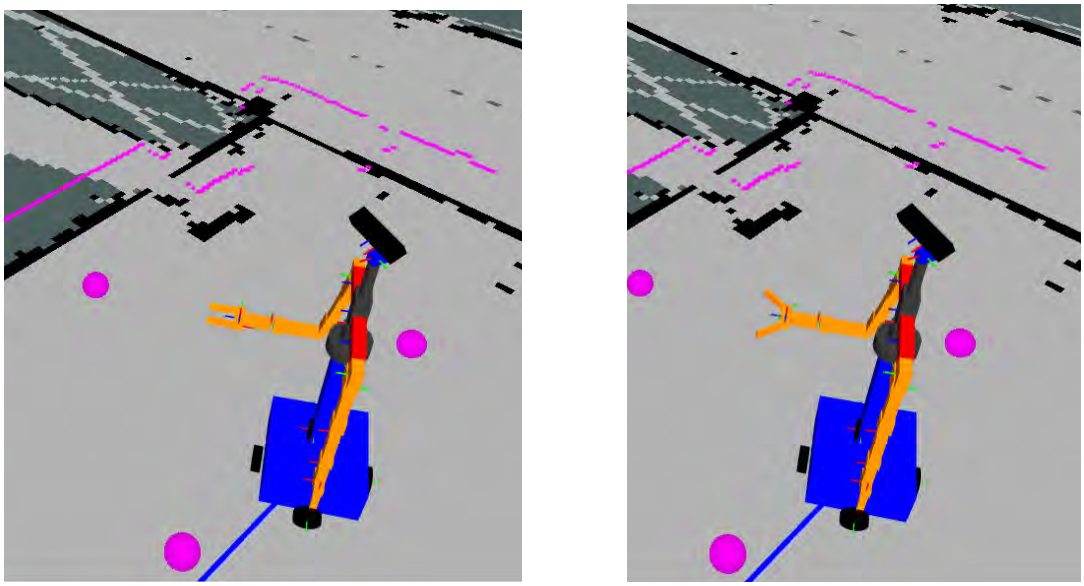
d)



e)



f)



g) h)

Figura 5-1. Plan ejecutado en el simulador gráfico.

Las figuras a) y b) muestran el modelo gráfico del robot Justina así como su posición al iniciar la prueba, también se muestra el mapa del ambiente donde debe interactuar el robot, en c) el robot se dirige a la "mesa" y mueve su manipulador para tomar un objeto, en d) el robot navega a la posición donde probablemente encontrará una persona, en e) y f) el robot comienza un ciclo de búsqueda rotando su cabeza y sobre si mismo para inspeccionar la escena y encontrar una persona, g) y h) al final el robot encontró a una persona, se acerca a ella, estira su brazo y abre la pinza que sujeta el objeto para dejarlo en manos de la persona.

Con este ejemplo se pude observar los movimientos que realiza el robot en la simulación, aunque hay que recalcar que en la simulación el robot no solo hace los movimientos, si no que también recibe los comandos de voz y el robot utiliza su generador de voz para hacer una interacción más natural con el usuario.

5.1.2 GPSR (General Purpose Service Robot)

El GPSR es una de las pruebas que se llevan a cabo en el RoboCup, en la categoría @HOME, y consiste en la evaluación de la interacción Humano-Robot y la integración de varias habilidades que debe poseer el robot como la navegación, reconocimiento de voz, reconocimiento de personas, reconocimiento de objetos, etc.

El robot debe resolver una tarea por petición de un humano. La petición no se conoce antes de iniciar la prueba por lo que no existen acciones predefinidas que el robot de antemano sepa que debe realizar. La tarea es un comando de voz generada aleatoriamente y esta compuesta de 3 acciones que involucran navegación, interacción Humano-Robot e interacción Humano-Objeto. El robot debe demostrar que ha reconocido las tres acciones repitiendo el comando que entendió y esperando por una confirmación. Si no puede entender el comando correctamente, el robot puede pedir que le repitan el comando o pedir más información.

La complejidad de esta prueba se enfoca en los siguientes aspectos:

- No existen acciones predefinidas para completar el comando.
- La complejidad en el reconocimiento de voz es elevada.

La prueba se lleva a cabo en un ambiente que simule las habitaciones, objetos, muebles y personas de una casa convencional. La figura 5-2 muestra el mapa del escenario donde se llevaron a cabo las pruebas del GPSR en Alemania 2016.

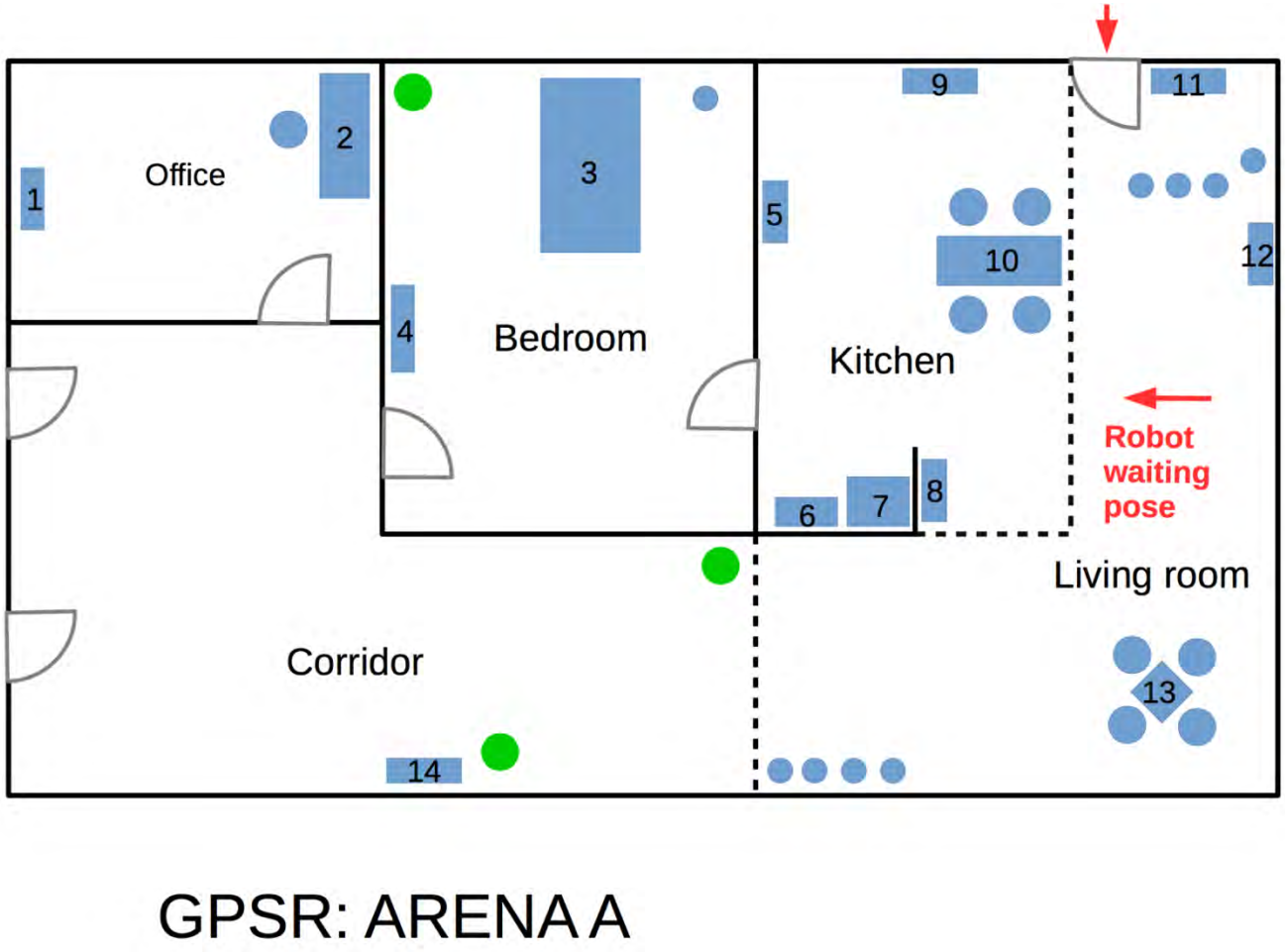


Figura 5-2 Mapa del escenario para la prueba GPSR, Robocup Leipzig 2016.

El robot debe esperar a que la puerta se abra para entrar a la arena y después debe navegar hasta la posición “Robot waiting pose” para recibir el comando de voz.

En la figura 5-3 se muestra una tabla describiendo los muebles de cada habitación, los objetos que se podrían encontrar dentro de cada habitación, y la indicación de si es posible realizar manipulación de objetos dentro de esa habitación.

	Área	Localización	Objeto asociado	Manipulación
1	Office	Drawer		Si
2	Office	Desk	Snacks	Si
3	Bedroom	Bed		No
4	Bedroom	Bedsides	Candies	Si
5	Kitchen	Bar		Si
6	Kitchen	Cupboard		Si
7	Kitchen	Sink	Containers	Si
8	Kitchen	Sideshelf	Food	Si
9	Kitchen	Bookcase	Drinks	Si
10	Kitchen	Dinning table		Si
11	Living room	TV stand		Si
12	Living room	Living shelf	Toiletries	Si
13	Living room	Living table		Si
14	Corridor	Cabinet		Si

Figura 5-3: Descripción del escenario de la Arena A.

La figura 5-4 muestra al robot de servicio Justina ejecutando la prueba GPSR. Se le ha solicitado navegar a la cocina y tomar el objeto *pringles*, para después entregarlo al usuario que hizo la petición.

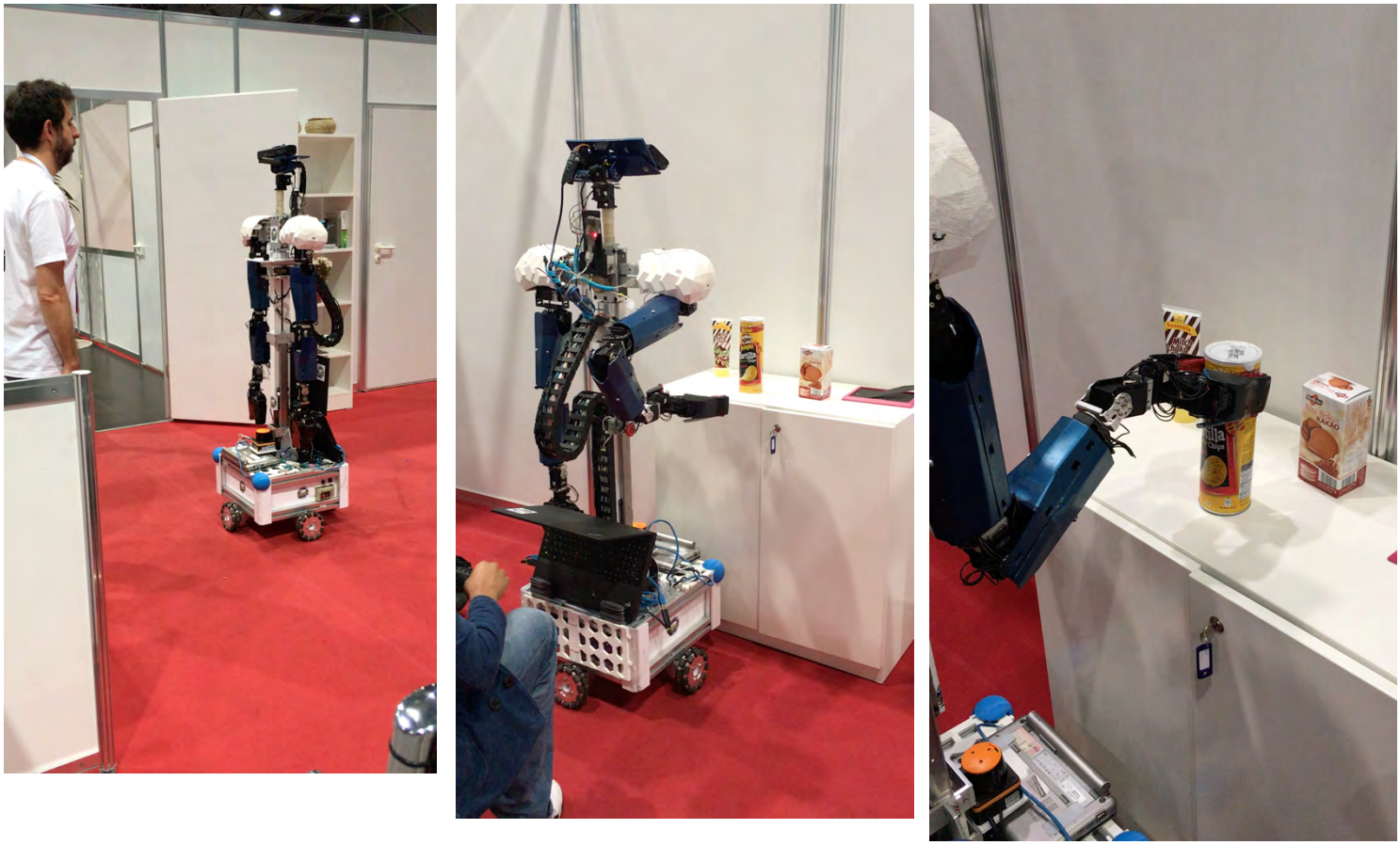


Figura 5-4: Prueba GPSR ejecutada por el robot de servicio Justina

Las figuras 5-5, 5-6 y 5-7 muestran parte del contenido de la bitácora de ejecución de esta prueba, en la cual se le a ordenado al robot buscar una persona en la cocina y contestar una pregunta. Para facilitar su lectura se omitieron líneas que no ofrecían información relevante a la planeación, como el proceso de eliminar hechos que ya no son utilizados y otros mensajes del proceso de descomposición de tareas realizado por el planeador.

```
--------- Command Speech ---------
name:cmd_speech
params:robotSpeech
GPSRTest.->First try to move
Navigation to arena
--------- Command Speech ---------
name:cmd_speech
params:robotSpeech
Response of wait for command:
Success:1
Args:ReadyToDevelopCommands
JustinaHRI.->Last reco speech: find a person in the kitchen and answer a question
JustinaHRI.->Received recognized speech: find a person in the kitchen and answer a question
--------- Command interpreter ---------
name:cmd_int
params:robot Interpreta
Response of interpreter:
Success:1
Args:find_a_person_in_the_kitchen_and_answer_a_question
--------- Command confirmation ---------
name:cmd_conf
params:find_a_person_in_the_kitchen_and_answer_a_question
------------ to_spech: ----------------- Do you want me find a person in the kitchen and answer a question
Response of confirmation:
Success:1
Args:plan-1930 2
JustinaHRI.->Last reco speech: robot yes
JustinaHRI.->Received recognized speech: robot yes
```

Figura 5-5. Bitácora de ejecución parte 1. Muestra los pasos de recepción de comando, interpretación y confirmación.

```
--------- Command get tasks ---------
name:cmd_task
params:robot Get Task
Response of get tasks:
Success:1
Args: user_speech find_person_in_room kitchen 1
--------- Command get tasks ---------
name:cmd_task
params:robot Get Task
Response of get tasks:
Success:1
Args: user_speech wait_for_user_instruction a_question 2
--------- Command get tasks ---------
name:cmd_task
params:robot Get Task
Response of get tasks:
Success:0
Args:No_Task
```

Figura 5-6. Bitácora de ejecución parte 2. Se muestra el proceso de creación del plan conformado por tareas de alto nivel.

```
47 --------- Command Navigation ---------
48 name:goto
49 params:kitchen kitchen -3.55 -3.0 0.0
50 Navigation to kitchen
51 --------- Command find a object ---------
52 name:find_object
53 params:person
54 find: person
55 JustinaVision.->Starting face recognition old.
56 Find a person
57 Move base
58 currAngleTurn:0
59 Sync move head start
60 Head goal:-0.785398
61 Sync move head end
62 JustinaVision.->Starting face recognition without id
63 recognized:0
64 Face centroid:0,0,0
65 End turnAndRecognizeFace
66 CentroidFace:0,0,0)
67 JustinaVision.->Stopping face recognition.
68 I have not found a person
69 JustinaVision.->Starting face recognition without id
70 recognized:1
71 Face centroid:1.3,5.5,0.1
72 Sync move head start
73 Head goal:0.785398
74 Sync move head end
75 JustinaVision.->Stopping face recognition.
76 I found a person
77 --------- Command Navigation ---------
78 name:goto
79 params:person person 1 1 1
80 --------- Command answer a question ---------
81 name:answer
82 params:question a_question
83 JustinaHRI.->Last reco speech: how tall are you
84 JustinaHRI.->Received recognized speech: how tall are you
85 --------- Command Navigation ---------
86 name:goto
87 params:robot exitdoor 0 0 0
88 Navigation to exitdoor
89 --------- Command Speech ---------
90 name:cmd_speech
91 params:robotSpeech
```

Figura 5-7. Bitácora de ejecución parte 3. Muestra la ejecución del plan.

En esta prueba deliberadamente se envío al robot a una localización donde no habían personas, en este caso la cocina. En la línea 68 de la bitácora de ejecución se puede observar que el robot no fue capaz de reconocer a ninguna persona dentro de la habitación, por lo que en un nuevo intento volvió a analizar el ambiente en busca de una persona, el robot realizo 3 intentos más en busca de una persona sin éxito, por simplicidad se omitieron la mayoría de intentos fallidos, hasta que al cuarto intento una persona entro en la habitación y el robot la reconoció, línea 76 de la bitácora. Cabe la posibilidad de que el plan fuera diseñado para poner en marcha otro tipo de acciones para que el robot corrigiera la situación desfavorable, aunque las limitaciones físicas y de respuesta de los sensores del robot afectan directamente en el tipo de planes que se pueden realizar.

5.1.3 Open Challenge

Es una prueba del RoboCup@HOME que consiste en incentivar a los equipos participantes a demostrar los resultados de sus investigaciones recientes y las mejores habilidades del robot participante. La prueba se enfoca en la demostración de nuevas aplicaciones, la interacción Humano-Robot y su valor científico.

En esta prueba se decidió mostrar la capacidad del robot de recordar información acerca de su entorno, antes y después de interactuar con el, así como su habilidad de poder verificar que es correcto lo que esta afirmando.

La presentación constó de los siguientes puntos:

1. El robot entra en escena, llega a una sala donde hay una mesa con objetos encima y dos personas (A y B) sentadas alrededor.
2. El Robot se presenta y notifica que esta listo para iniciar la prueba.
3. Las personas interactúan con el Robot con preguntas como:
    a. ¿Robot puedes describir lo que ves?
    b. ¿Dónde esta el objeto X?
    c. ¿Cuántas personas ves alrededor de la mesa?
4. La persona A realiza una petición al Robot de la forma:
    a. Robot dale el objeto X a la persona B
5. El robot pide que confirmen la petición.
6. La persona realiza la confirmación, pero le pide al robot que explique las acciones que va a realizar.
7. El robot explica el plan de acciones.
8. EL robot pide confirmación para llevar a cabo el plan.
9. La persona A permite que se lleve a cabo el plan, el plan se lleva a cabo con éxito.
10. La persona A pregunta por la ubicación del objeto X, El robot sabe que lo tiene la persona B, sin embargo la persona A le quito el objeto X a la persona B.
11. La persona A pide al robot que verifique su respuesta.
12. El robot se da cuenta que el objeto no lo tiene la persona B, y debe explicar porque no lo tiene.
13. Finaliza la presentación.

La figura 5-8 muestra al robot de servicio Justina llevando a cabo la prueba del Open Challenge. Sobre la mesa hay 4 objetos: coca, crema, una lata y pasta.

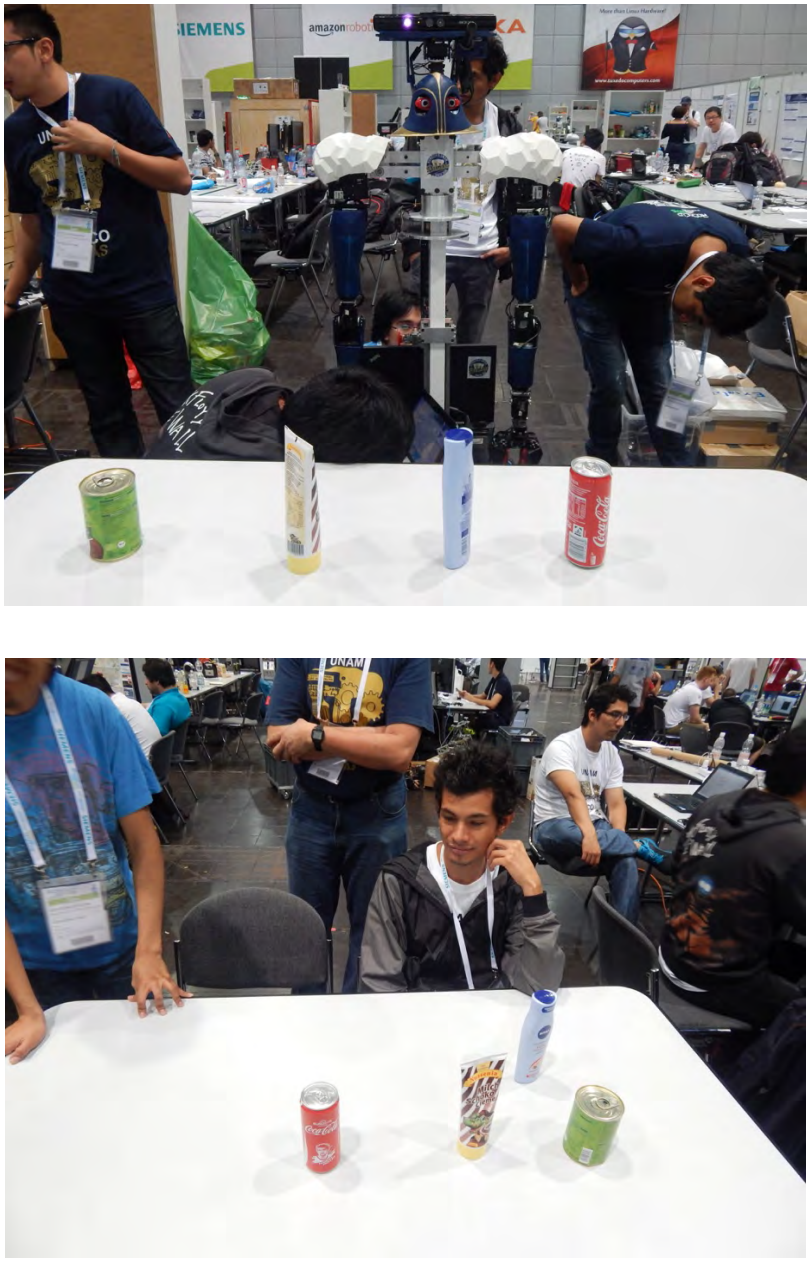


Figura 5-8: Prueba Open Challenge ejecutada por el robot de Servicio Justina

La prueba del Open Challenge representa todo un reto ya que el robot debe tener conocimiento total de lo que pasa en el mundo y como cambia a través de las acciones que va realizando; además, se le ha agregado un nivel de dificultad adicional al tener que comprobar que la información que tiene del mundo es correcta. En particular, se tienen muchos problemas a la hora de verificar que el objeto esta en posesión de la persona a la que le fue entregado, debido a que el método de detección de objetos esta basado en observar una imagen y dentro de esa imagen encontrar un plano, en este caso la mesa; eliminar todo lo que no este encima del plano; y, entonces, hacer el reconocimiento utilizando los segmentos de imagen que quedan (los objetos sobre la mesa) y el previo entrenamiento que se tiene de cada objeto. Ahora en el caso de hacer la detección del objeto en manos de una persona, se tiene el inconveniente de que el objeto no esta sobre un plano y puede estar parcial o totalmente cubierto por la mano de la persona que lo sostiene, haciendo muy difícil llevar a cabo la última parte del Open Challenge. Las figuras 5–9, 5–10, 5–11, 5–12 y 5–13 muestran el contenido de la bitácora de ejecución para la prueba del Open Challenge; al igual que en la bitácora del GPSR, se omitieron algunas líneas que no muestran información relevante acerca de la prueba.

La figura 5–9 muestra el inicio de la prueba del Open Challenge, en el cual el robot parte de una posición inicial preestablecida, se acerca a la mesa donde se encuentran los objetos e intenta alinearse con ella; en esta prueba después de varios intentos no consigue hacerlo, pero se continua con el siguiente estado. En el siguiente estado el robot esta listo para recibir comandos de voz y lo primero que recibe es inspeccionar el mundo en busca de objetos y personas, encuentra dos personas y dos objetos (línea 29 y 51 de la bitácora respectivamente), se observa que en un primer intento solo vio un objeto (línea 43), la coca, sin embargo con un movimiento lateral cambio el ángulo de visión y pudo detectar un objeto más.

```
--------- Command Init ---------
name:cmd_speech
params:robotSpeech
state:3
GPSRTest.->First try to move
Navigation to inspection
JustinaTasks.->Aligning with table. Moving head to 0 -0.9
JustinaTasks.->Requesting line to line_finder
JustinaVision.->Trying to find a straight line.
JustinaVision.->Cannot find line: cannot get point cloud :'(
JustinaTasks.->Cannot find line.
Align With table
Can not align with table.
--------- Command World ---------
name:cmd_world
params:world
JustinaHRI.->Last reco speech: tell me what do you see
JustinaHRI.->Received recognized speech: tell me what do you see
JustinaVision.->Starting face recognition.
Response of what do you see:
Success:1
Args:what_see_yes
JustinaVision.->Starting face recognition without id
JustinaNavigation.->Publishing goal lateral distance: 0.3
JustinaVision.->Starting face recognition without id
JustinaNavigation.->Publishing goal lateral distance: -0.3

peter times: 30
john times: 40
peterIzquierda times: 30
johnIzquierda times: 10
peterDerecha times: 0
johnDerecha times: 30

Find a object
Test object
JustinaVision.->Trying to detect objects...

Justina::Vision can't detect anything

Test object
JustinaVision.->Trying to detect objects...

    object detected: coke


JustinaNavigation.->Publishing goal lateral distance: 0.3

Test object
JustinaVision.->Trying to detect objects...

    object detected: coke
    object detected: cream
```

Figura 5-9. Bitácora Open Challenge parte 1. Inicio de la prueba.

```
--------- Command World ---------
name:cmd_world
params:world
JustinaHRI.->Last reco speech: describe what you known
JustinaHRI.->Received recognized speech: describe what you known
--------- Command Describe ---------
name:cmd_describe
params:peter izquierda john derecha
peter is in the right of john
john is in the left of peter
There are a coke in the table
There are a cream in the table

--------- Command World ---------
name:cmd_world
params:world
JustinaHRI.->Last reco speech: where is the soup
JustinaHRI.->Received recognized speech: where is the soup
--------- Command Where ---------
name:cmd_where
params:soup nil
The object soup is not in the room

--------- Command World ---------
name:cmd_world
params:world
JustinaHRI.->Last reco speech: where is the coke
JustinaHRI.->Received recognized speech: where is the coke
--------- Command Where ---------
name:cmd_where
params:stevia nil
The object coke is on the table

--------- Command World ---------
name:cmd_world
params:world
JustinaHRI.->Last reco speech: where is the cream
JustinaHRI.->Received recognized speech: where is the cream
--------- Command Where ---------
name:cmd_where
params:coffe nil
The object cream is on the table

```

Figura 5-10. Bitácora Open Challenge parte 2. Demostración del conocimiento del robot.

En la figura 5-10 las peticiones que se le hacen al robot son para saber que es lo que ha aprendido acerca del mundo donde se encuentra. En primer lugar, se le pide que indique que personas y que objetos sabe que están en la habitación, para lo cual el robot revisa su base de conocimientos y hace un resumen de lo que sabe (líneas 67 a 70).

Posteriormente se le pregunta individualmente por la localización de algunos objetos, como sabemos el robot reconoció *coke* y *cream* por lo que cuando se le pregunta por ellos contesta que están sobre la mesa (líneas 90 y 100). Ahora cuando al robot se le realiza la pregunta acerca de la localización de *soup*, el robot contesta que no se encuentra en la habitación (línea 80), lo cual es cierto por que no lo vio cuando se le indico inspeccionar la mesa. Aunque también es cierto que el objeto pudo estar sobre la mesa y el robot no lo vio, por estos casos el planeador de acciones tomaba acciones para poner al robot con diferentes ángulos de visión y no dejar de lado un objeto que estaba verdaderamente sobre la mesa, a pesar de eso en ocasiones sucedía que el robot no podía reconocer a un objeto, aquí el planeador de acciones optaba por seguir con el plan ya que hacer más movimientos resultaba en un desperdicio de tiempo y aun así en muchas ocasiones no se conseguía el resultado favorable que se hubiera esperado.

En la figura 5-11 se puede observar que al robot se le hace una petición del tipo de comandos del GPSR, el comando en concreto es tomar el objeto *coke* y entregarlo a John. Las acciones a partir de aquí son similares a las efectuadas en el GPSR, el usuario da el comando , línea 113, el robot pide una confirmación, línea 124, y el usuario responde afirmativamente. Ahora antes de ejecutar el plan el robot explica las acciones que tiene planeadas para terminar con éxito el comando solicitado, líneas 131 a 145.

```
101
102 --------- Command World ---------
103 name:cmd_world
104 params:world
105 JustinaHRI.->Last reco speech: could you take my order
106 JustinaHRI.->Received recognized speech: could you take my order
107 --------- Take order ---------
108 name:cmd_order
109 params:take_order
110 Response of wait for command:
111 Success:1
112 Args:ReadyToDevelopCommands
113 JustinaHRI.->Last reco speech: take the coke and deliver it to john
114 JustinaHRI.->Received recognized speech: take the coke and deliver it to john
115 --------- Command interpreter ---------
116 name:cmd_int
117 params:robot Interpreta
118 Response of interpreter:
119 Success:1
120 Args:take_the_coke_and_deliver_it_to_john
121 --------- Command confirmation ---------
122 name:cmd_conf
123 params:take_the_coke_and_deliver_it_to_john
124 ------------- to_spech: ------------------ Do you want me take the coke and deliver it to john
125 Response of confirmation:
126 Success:1
127 Args:plan-1615 3
128 JustinaHRI.->Last reco speech: robot yes
129 JustinaHRI.->Received recognized speech: robot yes
130
131 --------- Explain the plan ---------
132 name:cmd_explain
133 params:robot explain task
134 Response of explain plan:
135 Success:1
136 Args: user_speech get_object coke inspection 1
137 Response of explain plan:
138 Success:1
139 Args: user_speech find_person_in_room john inspection 2
140 Response of explain plan:
141 Success:1
142 Args: user_speech handover_object coke 3
143 Response of explain plan:
144 Success:0
145 Args:No_Tasks
```

Figura 5-11. Bitácora Open Challenge parte 3. Comando de voz y explicación del plan a realizar.

```
147 --------- Command Ask for ---------
148 name:ask_for
149 params:coke inspection
150 --------- Command Status object ---------
151 name:status_object
152 params:open_table
153 JustinaTasks.->Aligning with table. Moving head to 0 -0.9
154 JustinaTasks.->Requesting line to line_finder
155 JustinaVision.->Trying to find a straight line.
156 JustinaVision.->Cannot find line: cannot get point cloud :'(
157 JustinaTasks.->Cannot find line.
158 Align With table
159 Can not align with table.
160 --------- Command find a object ---------
161 name:find_object
162 params:coke
163 find: coke
164 Find a object coke
165 JustinaVision.->Trying to detect objects...
166
167 object detected: coke
168
169 --------- Command Move actuator ---------
170 name:move_actuator
171 params:soup 0 0 0
172 Move actuator coke
173 right arm
174 JustinaTasks.->Moving to a good-pose for grasping objects with right arm
175 JustinaTasks.->ObjToGrasp:   0  0  0
176 JustinaTasks.->Adjusting with frontal=-0.4 lateral=0.235 and vertical=-0.618
177 JustinaNavigation.->Publishing goal lateral distance: 0.235
178 JustinaTasks.->Moving right arm to 1.025  -0.35832  -0.451407
179 --------- Command grasp ---------
180 name:grab
181 params:manipulator coke
```

Figura 5-12. Bitácora Open Challenge parte 4. Ejecución del plan parte 1.

La primera parte de la ejecución consiste en alinearse con la mesa lo cual no se vuelve a conseguir, línea 153, después localizar el objeto *coke*, línea 164, luego mover el brazo a la posición del objeto, línea 174, y cerrar la pinza para tomar el objeto, línea 181.

```
182 |
183 --------- Command find a object ---------
184 name:find_object
185 params:specific john
186 find: specific
187 JustinaVision.->Starting face recognition old.
188 Find a person john
189 Move base
190 currAngleTurn:0
191 Sync move head start
192 Head goal:-0.785398
193 Sync move head end
194 JustinaVision.->Starting face recognition of id: john
195 recognized:1
196 Face centroid:0,0,0
197 End turnAndRecognizeFace
198 CentroidFace:0,0,0)
199 JustinaVision.->Stopping face recognition.
200 I have not found a person john
201 --------- Command Move actuator ---------
202 name:move_actuator
203 params:person -0.5 0.0 0.0
204 Move actuator person
205 right arm
206 JustinaTasks.->Moving to a good-pose for grasping objects with right arm
207 JustinaTasks.->ObjToGrasp:   -0.5  0  0
208 JustinaTasks.->Adjusting with frontal=-0.9 lateral=0.235 and vertical=-0.648
209 JustinaNavigation.->Publishing goal lateral distance: 0.235
210 JustinaTasks.->Moving right arm to 1.5194  0.778079  -0.488678
211 --------- Command Drop ---------
212 name:drop
213 params:person coke
214 --------- Command Navigation ---------
215 name:goto
216 params:robot exitdoor 0 0 0
217 Navigation to exitdoor
218 inspection
```

Figura 5-13. Bitácora Open Challenge parte 5. Ejecución del plan parte 2 y fin de la prueba.

La segunda parte de la ejecución consiste en que una vez el objeto esta en la pinza del robot se inicia el reconocimiento de personas, en este caso el robot busca a una persona en específico: John, línea 194, a continuación el robot se acerca a John, luego mueve su brazo hacia una posición donde el usuario pueda tomar el objeto, línea 206, finalmente el robot le advierte al usuario que soltara el objeto y que él debe sujetarlo para que no caiga al suelo, línea 213.

Y la prueba termina con el robot saliendo de la habitación. Es cierto que falta una parte del Open Challenge la cual consiste en verificar que el usuario en efecto recibió y aun esta en posesión del objeto, sin embargo por los problemas mencionados al inicio de este capítulo fue muy difícil por no decir que imposible realizar esa parte de la prueba, hay muchas cosas que el planeador debe tomar e cuenta para manejar los errores en esta parte de la prueba. En primer lugar si no detecta el objeto ¿que es lo que el robot debería hacer? no hay plano, por más que se cambie el ángulo de visión la probabilidad de que encuentre el objeto es muy baja. En segundo lugar el objeto puede estar total o parcialmente oculto detrás de la mano del usuario, una opción seria que el usuario colocara el objeto encima de la palma de su mano extendida de esta forma el objeto estaría visible y la palma de la mano actuaria como el plano necesario para el reconocimiento, y aun así no es nada confiable que la palma de la mano actué como un plano, el cuerpo humano tiende a estar en movimiento y por más que el usuario extienda bien la palma de su mano el usuario debe mantenerla en una postura idónea para que pueda pasar por un plano, se opto por no llevar esta idea a cabo ya que no resultaría muy natural hacer eso para facilitarle las cosas al robot, además para el usuario sería muy tedioso tener que permanecer en una postura cada vez que el robot quiera verificar donde se halla el objeto. En tercer lugar ¿qué sucede si el usuario realmente ya no tiene el objeto en su posesión? Resultaría inútil tratar de hacer las correcciones propuestas anteriormente para una persona que no tiene nada en las manos, lo que resultaría en perdida de tiempo para los usuarios y perdida de recursos del robot. Así que al termino de esta tesis aun se siguen buscando opciones para completar la parte faltante de la prueba del Open Challenge.

5.2 Evaluación del rendimiento del planeador de acciones

La evaluación del planeador de acciones se llevó a cabo comparando los resultados obtenidos en las simulaciones contra los resultados obtenidos en el robot real. Aunque en el laboratorio de Bio-Robótica existen antiguos motores de planeación, realizar una comparación entre estos y el motor desarrollado en esta tesis no resultaría relevante ya que los motores de planeación desarrollados anteriormente no estaban preparados para realizar las pruebas que se abordan en esta tesis.

La evaluación consistió en realizar 100 pruebas en las que cada una de ellas consistía de un comando generado aleatoriamente. Una lista completa de comandos generados aleatoriamente se pueden consultar en el apéndice D. Los comandos pueden diferir mucho entre si en el número de acciones que se deben realizar, sin embargo en la mayoría de los comandos se realizan las siguientes acciones básicas: navegación, reconocimiento de objetos, reconocimiento de personas, tomar objetos, soltar objetos y reconocimiento de voz. En la tabla 5-14 se lleva un registro donde se puede observar que para cada acción se lleva la cuenta de las veces que fue exitosa la acción y de las veces que fue fracaso; así como el cociente de éxito que consiste en dividir el total de acciones exitosas con el robot real entre el número de veces exitosas usando el simulador.

La gráfica 5-15 muestra la relación entre las acciones exitosas con el robot real y usando la simulación. Como se puede observar el éxito de las acciones con el robot real en su mayoría rebasa el 50% comparado con el éxito obtenido en la simulación. Las acciones con mayor índice de fracaso resultan ser las correspondientes a tomar y dejar objetos, porque dependen en gran medida del éxito de otras acciones como la navegación y la detección de objetos; si estas acciones fallan, es muy probable o casi un hecho que las acciones para tomar o dejar objetos fracase.

Acciones	Exitos		Fracasos		Real/Simulación
	Simulación	Real	Simulación	Real	
Navegacion	80	75	20	25	0.94
Reconocimiento de objetos	83	59	17	41	0.71
Reconocimiento de rostros	79	72	21	28	0.91
Tomar objetos	88	37	12	63	0.42
Depositar objetos	92	42	8	58	0.46
Reconocimiento de voz	95	82	5	18	0.86

Tabla 5–14. Registro de éxitos y fracasos para cien pruebas

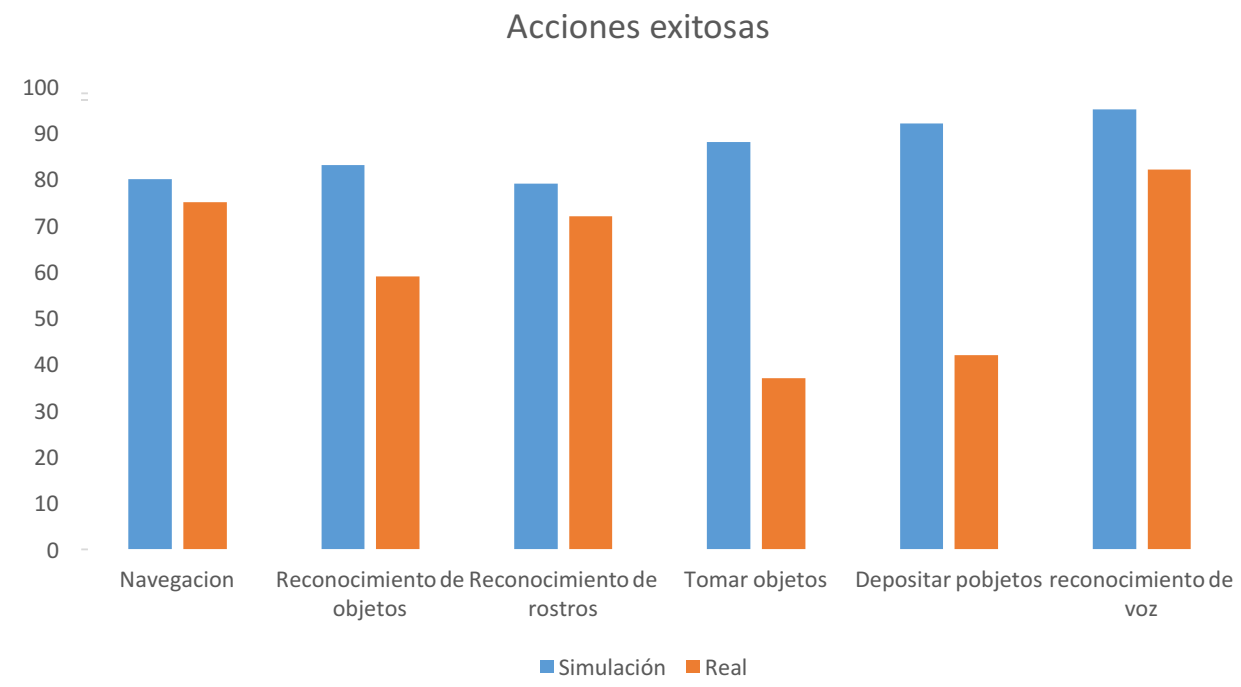


Figura 5–15. Gráfica de acciones exitosas simulación vs real.

Dejando de lado que las acciones para tomar o dejar objetos son dependientes de otras acciones, debemos hacer énfasis en que las primeras tienen sus propias dificultades para ser llevadas a cabo. Estas dificultades recaen en hechos que no tienen que ver directamente con la planeación, ya que para realizar la acción de tomar un objeto se deben mover con cuidado los motores del brazo para llegar a la posición correcta del objeto y una vez hecho esto se deben controlar los motores para cerrar la pinza que tiene el robot por mano y tomar el objeto, este procedimiento es similar para la acción de soltar objetos. En muchas ocasiones el robot no puede llegar a la posición correcta del objeto por que la manipulación mecánica de los motores no llega a ser tan precisa; el robot piensa que ha llegado a la posición correcta pero no es así. Por otra parte, cuando el robot logra exitosamente tomar el objeto a veces la fuerza de los motores no soporta el peso del objeto y lo dejan caer, lo cual se puede deber a que la batería del robot no provee el suficiente voltaje o a que el objeto en verdad es muy pesado. Que la batería del robot no provea el voltaje suficiente ha sido un gran problema ya que en las observaciones hechas la batería debería estar siempre cargada a tope por que aunque tuviera un poco menos del nivel total de carga los motores del brazo presentaban esos problemas.

Respecto a la navegación, el problema más común que se presentó es que el robot no podía calcular la ruta hacia su nuevo destino debido a que caía en un lugar donde de cualquier forma que trazara la ruta siempre encontraba un obstáculo que no lo dejaba avanzar. Así que aunque con el planeador de acciones hiciéramos varios intentos para completar la navegación esta jamás se podría llevar a cabo y por consecuencia las acciones siguientes fallarían. Para evitar el problema también se pensó en hacer navegar al robot un poco para salir del área con obstáculos y entonces trazar la ruta hacia su destino, sin embargo fue complicado decidir cuanta distancia tendría que moverse y en que dirección debía hacerlo, ya que haciendo un ligero movimiento el robot no saldría del área con obstáculos y haciendo movimientos más bruscos el robot podría chocar con los mismos obstáculos.

También hay que mencionar que el área de navegación afectaba el rendimiento del robot, en lugares muy pequeños el robot suele moverse con mayor precisión mientras que en lugares muy amplios el robot con frecuencia perdía la noción de su ubicación; por ejemplo si se encontraba en la sala y se le pedía ir a la oficina por más que se acercaba a la oficina seguía creyendo que se encontraba en la sala. Cuando esto sucedía no había forma de contrarrestar la falla con una nueva re planeación ya que los sensores seguían mostrando las mismas lecturas y era imposible reubicar de buena forma al robot.

El reconocimiento de objetos y personas presentó la mayoría de sus fracasos debido a la navegación, si el robot quedaba muy lejos de su destino no podía hacer mucho por reconocer bien lo que se le solicitaba. En general estas acciones trabajaban bien ya que se tiene un entrenamiento previo y robusto de objetos y rostros de personas, por supuesto el reconocimiento podía fallar por lo que el planeador de acciones tomaba medidas y añadía acciones para mover al robot en otra posición y ver la escena desde distintos ángulos, de esta manera se solventó el problema.

Por otra parte el reconocimiento de voz también presentó un gran rendimiento, las principales fallas se presentaron en un ambiente con mucha gente hablando y sonidos de distintos tipos, pero añadiendo un supresor de ruido al micrófono esto no representaba un gran problema. La única forma en que fallaba el reconocimiento era que el usuario no dijera el comando con la estructura correcta, y esto ocurría pocas veces, además el planeador tomaba medidas en caso de que el comando de voz que recibió fuera incorrecto, el robot solicitaba una confirmación antes de continuar, si el usuario no confirmaba el comando, el robot solicitaba de forma amable que le repitieran el comando correcto.

5.3 Observaciones

Todas las reglas para el ciclo de ejecución contienen un conjunto de condiciones iniciales mínimas, que restringen o habilitan dichas reglas dependiendo las circunstancias del entorno y en que estado del ciclo de ejecución se encuentre el robot. El hecho de que una regla este activa significa que las condiciones se han cumplido para activarla. Habrá que observar que estas condiciones en realidad son hechos, por lo que el interprete de CLIPS debe realizar una búsqueda en todas las reglas registradas para detectar que regla se activa y cual no.

Además, el control de ejecución de una tarea en la mayoría de los casos se determina por hechos que reflejan el estado del mundo, aunque también por hechos que no alteran el estado del mundo como lo es esperar por una confirmación o anunciar que se esta listo para recibir comandos, estos hechos funcionan como banderas que indican al planeador en que paso de la etapa de ejecución se encuentra y cual es la siguiente acción a realizar. En realidad el planeador de acciones realiza una búsqueda *greedy*.

El utilizar los hechos como condiciones en las reglas para controlar el flujo de ejecución es un tanto difícil de lograr. Si no se tiene cuidado se pueden dar casos en los que se habiliten dos reglas simultáneamente que no deberían estarlo, de modo que el flujo de ejecución podría tomar un camino distinto al esperado y por lo tanto arrojar resultados poco satisfactorios. De esta forma es muy importante mantener en las reglas una descripción detallada de las condiciones necesarias para su activación.

Cuando un plan aumenta de complejidad es más probable cometer errores al escribir el código en CLIPS, esto es debido a que hay que tener especial cuidado en no repetir nombres de reglas, que dos o más reglas tengan las mismas condiciones para su activación, y eliminar hechos que ya no son útiles para el flujo del ciclo de ejecución.

CAPÍTULO 6

CONCLUSIONES

De a cuerdo a los objetivos planteados para el desarrollo de esta tesis, los logros alcanzados son los siguientes:

- Se logró crear una interfaz de comunicación entre el *planeador de acciones* y el ambiente gráfico de ROS para la simulación de la ejecución de los planes.
- La integración del *planeador de acciones* con el robot *Justina* se llevó a cabo con éxito.
- Se logró interpretar y representar los comandos de voz para la elaboración de planes cuya estructura contiene las acciones necesarias para completar la petición.
- Se elaboraron reglas para el *planeador de acciones* que permiten supervisar situaciones en las que alguna acción del plan ha fracasado. Estas reglas determinaran si la ejecución del plan debe terminar o si es posible seguir con la ejecución de la siguiente acción.
- El *planeador de acciones* descrito en este trabajo se utilizó en la competencia Robocup@Home celebrada en Leipzig, Alemania 2016, en la prueba GPSR, mostrando buen desempeño en reconocimiento de voz y navegación.
- Finalmente se llevó a cabo la implementación del *planeador de acciones* para la prueba Open Challenge.

Con los objetivos alcanzados en esta tesis se puede dar respuesta a la pregunta que se realizó en la hipótesis y decir que al entender y modelar el conocimiento humano para la creación del *planeador de acciones*, el robot *Justina* fue capaz de llevar a cabo tareas diarias del hogar y mostrar un comportamiento en sus acciones comparable a la de una persona, lo que finalmente se traduce en una interacción humano-robot más natural.

Por otra parte debemos tener en mente que existen ciertos aspectos que se deben considerar para la elección de un planeador. Dichos aspectos implican la naturaleza subjetiva en la descripción de los planes, las acciones y el ambiente donde se desarrollaran. Debido a la gran variedad de escenarios que se pueden presentar, no podemos decir que un planeador es absolutamente mejor que otro; si no que debemos distinguir y ponderar que características ofrece cada uno para que al final elijamos el planeador que más se adapte a la situación y problema a resolver. En este capítulo se describirán las implicaciones de utilizar el planeador presentado en capítulos anteriores en base a las observaciones obtenidas al presentar los resultados.

La complejidad del planeador de acciones radica en la cantidad de criterios que se deben especificar en los casos en los que una tarea ha fracasado o ha tenido éxito; entre más condiciones se quieran controlar más reglas se deben diseñar para encontrar una solución al problema de planeación. Además, las reglas no deben interferir con el flujo de ejecución, es decir, el control de estas reglas no permitirá que la ejecución quede estancada. Para esto, se notificará que no se puede seguir realizando la acción que implica la regla y se continuará con el intento de realizar la siguiente acción en el plan.

El que los planes estén estructurados en tareas nos permite dividir la solución de un problema complejo en soluciones más específicas y menos complicadas de acuerdo a cada tarea. Así mismo las tareas son divididas en subtareas, por lo que el control de ejecución no regresará a la tarea padre hasta que se reciba una respuesta de éxito o fracaso de todas las subtareas que conforman a la tarea padre. Cuando la tarea padre recupere el control de ejecución se evaluarán los resultados obtenidos en las subtareas y se decidirá que acciones se proceden a realizar.

Pueden ocurrir escenarios en los que una tarea que es de mayor jerarquía deba detener una subtarea (de menor jerarquía), los motivos son que sus criterios de éxito o fracaso se han cumplido. El planeador por si solo no podría darse cuenta por lo que existen reglas independientes a las reglas de planeación de la tareas que permiten detectar el estado de las subtareas y decidir el momento para detenerla.

También se pudo observar que en este planeador es necesario tener una descripción correcta del estado del mundo y las acciones que recaen sobre él para conseguir un plan adecuado, y que las acciones se han generalizado en un número limitado de posibles tareas que el robot tendría que efectuar para terminar con éxito el plan. Por lo que sería complicado representar una acción particular que no ha sido contemplada en las tareas que puede manejar el planeador, por lo que seria necesario agregar nuevas reglas y condiciones que activen y desactiven la acción cuando es debido. Si no se tiene el debido cuidado con las condiciones de activación de las reglas estás pueden tener comportamientos fuera de lo esperado y activarse cuando no corresponde, o viceversa, no activarse cuando se requiere, o también detener por completo el ciclo de ejecución.

Realizar modificaciones a los planes resulta difícil, ya que también implica modificar las condiciones de activación de todas o casi todas las reglas que implican la tarea que se requiere modificar.

Sin embargo, cuando ya se tienen bien definidas las tareas y acciones que se realizarán, se puede en cierta forma, elaborar una máquina de estados más simple para inicializar los procesos en CLIPS y levantar las llamadas a servicios; de esta forma, modificar la máquina de estados principal del planeador de acciones no resulta tan complejo y no se alteran las reglas de creación de planes o las reglas correspondientes a la realización de alguna tarea.

El sistema experto utilizado propicia un diseño de planes basado en objetivos mas que en procedimientos, lo que significa que la robustez de la planificación será proporcional al número de escenarios distintos considerados para el desarrollo de tareas por parte del robot. La especificación de condiciones para cada regla permite considerar los distintos escenarios y elaborar un plan según sea conveniente.

Es importante decir que implementar muchas de las funcionalidades que realiza la máquina de inferencias que incluye el intérprete de CLIPS en un planeador procedural sería una tarea compleja ya que se tendría que hacer uso de técnicas

especializadas como el uso de programación multi-hilo, para sincronizar y coordinar de alguna forma todos los casos o condiciones que implica la planeación de acciones, o bien implementar un mecanismo de búsqueda similar al del intérprete de CLIPS.

6.1 Trabajo futuro

El planeador de acciones se realizó específicamente para llevar a cabo la prueba GPSR, por lo que seria importante añadir funciones más generales que permitan hacer uso de la estructura base de este planeador y poder utilizarlo en otro tipo de pruebas; por ejemplo, la estructura base del planeador fue utilizada para la Prueba Open Challenge lo que resultó en una tarea con un grado de dificultad grande ya que se tuvo que modificar la maquina de estados principal del flujo de ejecución para que se adaptara a la prueba.

Así mismo, el cambiar de prueba implica también cambiar la estructura de los comandos de voz, los cuales pueden variar mucho de prueba a prueba y el principal problema es que muchas veces el interprete arrojará resultados no deseados al no emparejar de buena forma el comando de voz con una secuencia de acciones aceptables. Por lo que es necesario encontrar otra forma de realizar la interpretación o agregar un modulo de interpretación diferente para cada prueba y así obtener buenos resultados para cada tipo de comando de voz. De cualquier forma el interprete necesitaría de una lógica de descripción con alguna ontología conveniente para mantener la representación adecuada para distintas pruebas.

Finalmente pensando en que en un mismo ambiente exista la posibilidad de que haya más de un robot interactuando con los usuarios, se podría ampliar el planeador a un sistema multi-agente, añadiendo un campo a las condiciones de las tareas para indicar a que robot le corresponde realizar las acciones necesarias para cumplir con éxito una tarea.

APÉNDICE A

GRAMÁTICA

En este apéndice se abordara la especificación de la gramática utilizada para generar y reconocer los comandos de voz. La gramática esta definida en un archivo XML que se explica a continuación.

```
<?xml version="1.0" encoding="utf-8"?>
<grammar version="1.0" xml:lang="es-ES" root="main"
xmlns="http://www.w3.org/2001/06/grammar">
    <rule id="main" scope="public">
     <one-of>
          <item>
               <ruleref uri="#complex" />
          </item>
          <item>
               <ruleref uri="#_questions" />
          </item>
          <item>
               <ruleref uri="#confirmation_polite"/>
          </item>
     </one-of>
    </rule>
```

Comandos (oraciones complejas)

Preguntas

Oraciones de confirmación

Figura A-1. Especificación de la gramática para reconocimiento de comandos de voz.

La regla principal se conforma de 3 reglas que se expandirán para reconocer preguntas, oraciones de confirmación, y los comandos que debe realizar el robot. La figura A-2 muestra como se conforma la regla “complex”.

1. Reconoce un comando que solicite obtener un objeto y entregarlo a una persona

```
<item>
     <ruleref uri="#getobj" /> and <ruleref uri="#deliver" /></item>
```

2. Reconoce un comando que solicite navegar a una habitación y encontrar un objeto

```
<item>
     <ruleref uri="#goroom" /> and <ruleref uri="#findobj" /></item>
```

3. Reconoce un comando que solicite navegar a una habitación, encontrar una persona y una acción adicional

```
<item>
     <ruleref uri="#goroom" />, <ruleref uri="#findprs" /> and <ruleref
uri="#complex_0" /></item>
```

4. Reconoce un comando que solicite la combinación de dos tareas complejas

```
<item>
     <ruleref uri="#complex_1" /> and <ruleref uri="#complex_2"/></item>
```

Figura A-2. Expansión y componentes de la regla complex.

Las reglas complex_1 y complex_2 forman oraciones más elaboradas las cuales contienen acciones específicas que se muestran en la figura A-3.

```
<rule id="complex_1" scope="private">
     <one-of>
          <item>
               <ruleref uri="#findprsat" />
          </item>
          <item>
               <ruleref uri="#fndclprsat" />
          </item>
     </one-of>
    </rule>

<rule id="complex_2" scope="private">
     <one-of>
          <item>
               <ruleref uri="#talk" />
          </item>
          <item>
               <ruleref uri="#follow" />
          </item>
     </one-of>
    </rule>
```

Acciones para encontrar una persona en un lugar específico.

Acciones para hablar o seguir a una persona

Figura A-3. Expansión y componentes de las reglas complex_1 y complex_2.

Ahora que se han desglosado las reglas generales de la gramática pasaremos a revisar las reglas que definen acciones o comandos en específico, las siguientes figuras describirán las acciones: getobj, deliver, goroom, findobj, findprs, findprsat, findclprsat, talk and follow.

```
<rule id="getobj" scope="private">
     <item>
          <ruleref uri="#vbtake" />
          the <ruleref uri="#_kobjects" />
          from the <ruleref uri="#_placements" />
     </item>
</rule>


<rule id="deliver" scope="private">
     <item>
          <ruleref uri="#vbdeliver" />
          it to <ruleref uri="#target" />
          </item>
    </rule>

<rule id="target" scope="private">
     <one-of>
          <item>me</item>
          <item>
               <ruleref uri="#_names" /> <ruleref uri="#target_0" />
               the <ruleref uri="#_rooms" /></item>
          <item>the <ruleref uri="#_placements" /></item>
     </one-of>
</rule>


<rule id="goroom" scope="private">
     <item>
          <ruleref uri="#vbgoto" /> the <ruleref uri="#_rooms"/>
    </item>
</rule>
```

La acción obtener objeto se compone del verbo tomar, el nombre del objeto y su localización

La acción soltar objeto se compone del verbo dejar seguido del objetivo donde se desea dejar el objeto

El objetivo puede ser el usuario, otra persona en diferente habitación o un mueble

La acción ir a una habitación se compone del verbo navegar y el nombre de la habitación

Figura A-4. Acciones obtener objeto, soltar objeto y navegar.

```
<rule id="findobj" scope="private">
     <item>
          <ruleref uri="#vbfind" />
          the <ruleref uri="#_kobjects"/>
    </item>
</rule>


<rule id="findprs" scope="private">
     <item>
          <ruleref uri="#vbfind" /> a person
    </item>
</rule>


<rule id="findprsat" scope="private">
     <item>
          <ruleref uri="#vbfind" />
          a person in the
          <ruleref uri="#partyroom" />
    </item>
</rule>


<rule id="fndclprsat" scope="private">
     <item>
          <ruleref uri="#vbfind" />
          the calling person in the
          <ruleref uri="#partyroom" />
    </item>
</rule>


<rule id="partyroom" scope="private">
     <item>
          <ruleref uri="#_rooms" />
     </item>
</rule>
```

La acción encontrar objeto consta del verbo buscar y el nombre del objeto.

La acción encontrar a una persona no especificada consta del verbo buscar

La acción encontrar a una persona en una habitación consta del verbo buscar y el nombre de la habitación

La acción encontrar a una persona específica en una habitación consta del verbo buscar y el nombre de la

Figura A-5. Acciones buscar objeto, buscar persona, buscar persona en habitación, buscar persona específica en habitación.

```
<rule id="follow" scope="private">
      <one-of>
            <item>
                  <ruleref uri="#vbfollow" /> her
            </item>
            <item>
                  <ruleref uri="#vbfollow" />
                  her to the
                  <ruleref uri="#_rooms" />
            </item>
      </one-of>
</rule>


<rule id="talk" scope="private">
      <one-of>
            <item>answer a question</item>
            <item>
                  <ruleref uri="#tell" />
            </item>
      </one-of>
</rule>

<rule id="tell" scope="private">
      <item>
            <ruleref uri="#vbspeak" />
            <ruleref uri="#tell_0" />
    </item>
</rule>

<rule id="tell_0" scope="private">
      <one-of>
            <item>
                  <ruleref uri="#time" />
            </item>
            <item>
                  <ruleref uri="#date" />
            </item>
            <item>
                  <ruleref uri="#name" />
            </item>
      </one-of>
    </rule>
```

La acción seguir se compone del verbo seguir y el nombre de la habitación hasta donde se va a seguir a la persona

La acción hablar se compone de una nueva acción “tell” o simplemente contestar una pregunta

La acción “tell” se compone del verbo decir y de la información que se dirá (la hora, el día o el nombre del robot)

Figura A-6. Acciones seguir y hablar con una persona.

Cuando al robot se le solicita interactuar con una persona existen ciertas reglas en la gramática que restringen las formas de interacción entre el robot y el usuario, por ejemplo el robot solo puede atender peticiones de lo forma: decir su nombre, decir el nombre de su equipo, decir que día es, que hora es, etc. En la figura A-7 se muestran todas las posibles formas de interacción según las reglas de la gramática.

```
<rule id="name" scope="private">
      <one-of>
            <item>your name</item>
            <item>the name of your team</item>
      </one-of>
</rule>


<rule id="time" scope="private">
      <one-of>
            <item>the time</item>
            <item>what time is it</item>
      </one-of>
</rule>


<rule id="date" scope="private">
      <one-of>
            <item>the date</item>
            <item>what day is
                        <ruleref uri="#day" /></item>
            <item>the day of the
                        <ruleref uri="#date_0" /></item>
      </one-of>
</rule>

<rule id="day" scope="private">
      <one-of>
            <item>today</item>
            <item>tomorrow</item>
      </one-of>
</rule>

<rule id="date_0" scope="private">
      <one-of>
            <item>month</item>
            <item>week</item>
      </one-of>
</rule>
```

1. El robot dice su nombre
2. El robot dice el nombre de su equipo

1. El robot dice el clima
2. El robot dice la hora

1. El robot dice la fecha
2. El robot dice que día es hoy o mañana
3. El robot dice que día de la semana es

Figura A-7. Formas de interacción robot-usuario.

En la gramática se listan los verbos para cada acción y también algunos sinónimos para contemplar un vocabulario más amplio. Los verbos y su representación en la gramática se muestran en la figura A-8.

```
<rule id="vbdeliver" scope="private">
     <one-of>
          <item>bring</item>
          <item>carry</item>
          <item>deliver</item>
          <item>take</item>
     </one-of>
</rule>

<rule id="vbfind" scope="private">
     <one-of>
          <item>find</item>
          <item>look for</item>
     </one-of>
</rule>

<rule id="vbfollow" scope="private">
     <item>follow</item>
</rule>

<rule id="vbgoto" scope="private">
     <one-of>
          <item>go to</item>
          <item>navigate to</item>
          <item>reach</item>
          <item>get into</item>
     </one-of>
</rule>

<rule id="vbtake" scope="private">
     <one-of>
          <item>take</item>
          <item>grasp</item>
          <item>get</item>
     </one-of>
</rule>

<rule id="vbspeak" scope="private">
     <one-of>
          <item>tell</item>
          <item>say</item>
          <item>speak</item>
     </one-of>
</rule>
```

Verbos para soltar un objeto

Verbos para buscar un objeto o una persona

Verbos para seguir a una persona

Verbos para navegación

Verbos para tomar objetos

Verbos para interactuar con una persona

Figura A-8. Representación de los verbos soltar, buscar, seguir, navegar, tomar y hablar en la gramática.

También se especifican en la gramática el nombre de las habitaciones que conforman el área por donde navegara el robot así como el nombre de los muebles u otros lugares donde el robot podrá tomar o dejar objetos. Dichos nombres se muestran en la figura A-9.

```
<rule id="_locations" scope="private">
     <one-of>

          <item>
               <ruleref uri="#_placements" />
          </item>
          <item>
               <ruleref uri="#_rooms" />
          </item>
     </one-of>
</rule>

<rule id="_placements" scope="private">
     <one-of>
          <item>dinner table</item>
          <item>kitchen table</item>
          <item>stove</item>
          <item>fridge</item>
          <item>bed</item>
          <item>side table</item>
          <item>sofa</item>
          <item>couch</item>
          <item>coffe table</item>
     </one-of>
</rule>

<rule id="_rooms" scope="private">
     <one-of>
          <item>dining room</item>
          <item>kitchen</item>
          <item>bedroom</item>
          <item>living room</item>
          <item>dining room</item>
          <item>kitchen</item>
          <item>bedroom</item>
          <item>living room</item>
     </one-of>
</rule>
```

Dos tipos de locaciones:
1. Muebles
2. Habitaciones

Varios muebles

Nombre de las habitaciones

Figura A-9. Representación de los muebles y habitaciones en la gramática.

El nombre de las personas con las que el robot puede interactuar también se especifican en la gramática y se muestran en la figura A-10, los nombres están divididos en dos categorías: hombre y mujer.

```
<rule id="_names" scope="private">
     <one-of>
          <item>
               <ruleref uri="#_males" />
          </item>
          <item>
               <ruleref uri="#_females" />
          </item>
     </one-of>
</rule>


<rule id="_males" scope="private">
     <one-of>
          <item>james</item>
          <item>robert</item>
          <item>arthur</item>
          <item>mike</item>
          <item>richi</item>
          <item>noah</item>
     </one-of>
</rule>


<rule id="_females" scope="private">
     <one-of>
          <item>susan</item>
          <item>mary jane</item>
          <item>gabrielle</item>
          <item>elsa</item>
     </one-of>
</rule>
```

Regla general para los nombres clasificados en dos categorías: hombre y mujer

Lista de hombres

Lista de mujeres

Figura A-10. Representación de los nombres de las personas en la gramática.

Así mismo es necesaria una lista con el nombre de los objetos con los cuales el robot podrá interactuar. La lista de objetos así como su representación en la gramática se muestra en la figura A-11.

```
<rule id="_objects" scope="private">
     <one-of>
          <item>grape juice</item>
          <item>coke</item>
          <item>cranberry juice</item>
          <item>nescafe latte</item>
          <item>pocky</item>
          <item>soup</item>
          <item>soap</item>
          <item>ajax</item>
          <item>shoe cleaner</item>
     </one-of>
</rule>
```

Lista de objetos

Figura A-11. Representación de los nombres de los objetos en la gramática.

Finalmente la gramática debe ser capaz de reconocer comandos de confirmación. La gramática de estos comandos se muestra en la figura A-12.

```
<rule id="confirmation_polite" scope="public">
     <one-of>
          <item>
               <ruleref uri="#CONFIRMATION" />
          </item>
     </one-of>
</rule>

<rule id="CONFIRMATION"  >
     <one-of>
          <item>robot yes</item>
          <item>robot no</item>
          <item>yes</item>
          <item>no</item>
     </one-of>
</rule>
```

Regla para recibir confirmaciones afirmativas o negativas

Figura A-12. Representación de las oraciones de confirmación.

APÉNDICE B

SISTEMA EXPERTO CLIPS

CLIPS (C Language Integrted Production System), es una herramienta para sistemas expertos originalmente desarrollada por la STB (Software Technology Branch), en el centro espacial Lyndon B. Johnson de la NASA. Desde su lanzamiento en 1986, CLIPS ha pasado por un continuo refinamiento y mejoramiento y fue diseñado para facilitar el desarrollo de software especializado en modelar el conocimiento humano. Se recomienda consultar la página http://clipsrules.sourceforge.net/, donde podrá encontrar la documentación y los manuales de referencia de CLIPS[25].

Un sistema experto es aquel que contiene información sobre un dominio particular, la cual se obtiene del conocimiento de una persona que se considera experta en la materia. Existen 3 formas para representar conocimiento en CLIPS:

- *Rules*: Destinadas principalmente para conocimiento heurístico basado en la experiencia.
- *Deffunctions* y *generic functions*: Se destinan principalmente para conocimiento procedural.
- *Object-oriented programming*: También se destina para conocimiento procedural. Soporta las cinco características presentes en la programación orientada a objetos: clases, manejador de mensajes, encapsulación, herencia y polimorfismo.

Los lenguajes procedurales comunes como C, Java, etc. son poco prácticos para representar y resolver problemas que modelen conocimiento humano, ya que implican el manejo de una gran cantidad de condiciones, por lo cual lenguajes de programación lógica como Prolog o CLIPS son una mejor opción para abordar dichos problemas. Los lenguajes de programación lógica facilitan la representación simbólica de problemas y su manipulación a través de mecanismos de unificación e inferencia.

Cabe mencionar que CLIPS esta diseñado para poder integrarse con otros lenguajes procedurales, es decir, CLIPS es una herramienta que puede ser llamada desde un lenguaje procedural, ejecutar las funciones solicitadas y regresar el control al programa que hizo la llamada. Por otra parte, el código procedural puede ser definido como funciones externas que se ejecutarán cuando CLIPS lo requiera y una vez que se complete la ejecución del código externo, el control regresará de a CLIPS.

Los intérpretes de los lenguajes de programación lógica emplean una máquina de inferencias, que consiste en algoritmos de búsqueda y unificación para encontrar correspondencias entre el estado del mundo, y los hechos y reglas

que se definieron en el sistema experto.

CLIPS se basa en reglas de encadenamiento hacia adelante (forward chaining), lo que le permite utilizar los hechos que conoce sobre la situación actual y su conocimiento sobre el mundo para activar las reglas correspondinetes y generar nuevos hechos.

B.1. Constructs

Distintos tipos de *constructs* son definidos en CLIPS, las más utilizadas en este trabajo de tesis fueron: deffacts, que sirve para especificar los hechos iniciales de un programa; deffunction, utilizada para definir funciones; deftemplate, que se usan en el diseño de templates que servirán para inicializar hechos estructurados; y defrule, que sirve para establecer las reglas del programa.

B.2. Hechos

Los hechos son una de las formas básicas de alto nivel para representar información en un sistema experto en CLIPS, así mismo, se definen como la unidad fundamental de información necesaria para el uso de reglas. Además cada hecho representa una pieza de información que se incluye en la lista de hechos, la cual permanecerá activa hasta que el programa termine de ejecutarse.

Los hechos se pueden añadir a la lista de hechos usando el comando *assert*, se pueden remover usando el comando *retract*, se pueden modificar usando el comando *modify*, o se pueden duplicar usando el comando *duplicate.*

Cuando un hecho es agregado o modificado en la lista de hechos, se le asigna un número entero único, que representa el índice de ese hecho. Para la asignación del índice la cuenta comienza en cero y se va incrementando en uno cada que un hecho se añade o modifica. Cuando el comando *reset* o *clear* se activa, la cuenta regresa a cero.

B.2.1. Hechos ordenados

Los hechos ordenados consisten en un símbolo seguido de una secuencia de cero o más campos separados por espacios y delimitados por un paréntesis abriendo a la izquierda, y un paréntesis cerrando a la derecha. El primer elemento de un hecho ordenado establece una “relación” con los elementos restantes del hecho ordenado. Por ejemplo, *(hijo_de rick isidore)* establece que isidore es el hijo de rick.

Algunos ejemplos de hechos ordenados son:

```
(objeto coke 0.0 1.0 0.0)
(objeto kitchen_table mueble 5.0 2.0 0.0)
(present-actor robot)
```

B.2.2. Hechos no ordenados

Los hechos ordenados codifican información posicionalmente, es decir, para acceder a la información el usuario debe conocer de que campos esta constituido el hecho y en que orden aparecen para extraer la información correcta. Los hechos no ordenados (non-ordered facts), o también llamados hechos estructurados (template facts) proveen al usuario la capacidad de abstraer la estructura de un hecho mediante la asignación de nombres a cada campo del hecho. De esta forma se accede a los campos de un hecho por su nombre y no por su posición.

Algunos ejemplos de hechos estructurados son:

```
(objeto (nombre soup) (pos_X 0.0) (pos_Y 0.0) (pos_Z 0.0) )

(objeto (nombre coke) (pos_X 0.0) (pos_Y 1.0) (pos_Z 0.0) )

(objeto (nombre kitchen_table) (pos_X 5.0) (pos_Y 2.0) (pos_Z 0.0) )

(present-actor (nombre robot) (zone entrance))
```

Los hechos estructurados permiten definir el nombre para un template en conjunto con cero o más campos que lo conformaran (slots). Los hechos estructurados se diferencian con los hechos ordenados en que no es necesario recordar ningún orden en particular, ni especificar todos los campos que conforman un hecho para poder: referenciarlo en una regla; o añadirlo, removerlo y modificarlo en la lista de hechos.

Los atributos de default permiten especificar un valor por default, mientras que los atributas de restricción limitan los valores que se pueden asignar, ya sea de tipo o de valor.

Un ejemplo de deftemplate:

```
(deftemplate plan

    (field nombre  (type SYMBOL)

    (field numero  (type NUMBER) (default 1))
```

```
        (multifield actions  (type SYMBOL))
        (field duration  (type NUMBER) (default 1))
        (field status  (type SYMBOL) (default inactive))
        (field statusTwo  (type SYMBOL) (default active))
    )
```

B.2.3. Hechos iniciales

Los hechos iniciales (deffacts) sirven para definir los hechos iniciales de un programa. Cuando la función reset se ejecuta se borrarán todos los hechos contenidos hasta ese momento en la lista de hechos y se añadirán todos los hechos contenidos en deffacts a la lista de hechos.

A continuación se muestra un ejemplo:

```
    (deffacts inicio "Hechos que inicializan los objetos en el mundo"
        (numero_de_objetos 2)
        (objeto (nombre soup) (pos_X 5.0) (pos_Z 1.0) )
        (objeto (nombre coke) (pos_Y 4.0))
    )
```

B.3. Reglas de CLIPS

Una regla es el método principal por medio del cual se puede representar conocimiento en CLIPS. Una regla es una colección de condiciones y acciones que se llevaran a cabo solo si todas las condiciones que la conforman se cumplen. El desarrollador de un sistema experto define las reglas que describen cómo resolver un problema y estas se ejecutan basándose en la existencia o la no existencia de hechos, para ello CLIPS provee una máquina de inferencias que devuelve las reglas a ejecutar en el estado actual del sistema.

Los elementos condicionales de una regla se especifican en el lado izquierdo (left hand side, LHS) de la regla. Pueden constar de hechos ordenados, hechos no ordenados, variables, etc. que se usan para restringir el conjunto de hechos que deben existir para activar la regla.

Las acciones se especifican en el lado derecho (rigt hand side, RHS) de la regla y consisten en: crear nuevos hechos, eliminar hechos que dejaron de ser ciertos en el mundo y modificar hechos. La creación, eliminación y modificación de hechos dará paso a que se habiliten otras reglas, gracias a lo cual podemos manipular el ciclo de ejecución del programa para que no se detenga.

El LHS y el RHS de una regla están separados por el símbolo "=>". Las acciones de una regla corresponden a cero o más funciones que se ejecutan de manera secuencial.

A continuación se muestra un ejemplo de la declaración de una regla:

```
(defrule task_find_specific_person_in_room
    ?f <- (task ?plan find_person_in_room ?person ?place ?step)
    ?f1 <- (item (name ?place))
    ?f2 <- (item (name ?person))
    =>
    (retract ?f)
    (printout t "Find Specific person in room" crlf)
    (assert (state (name ?plan) (number ?step)))
    (assert (cd-task (cd pfindspcperson) (actor robot)))
    (modify ?f1 (status nil))
    (modify ?f2 (status nil))
)
```

APÉNDICE C

ROS-PYCLIPS

Para realizar la conexión entre ros y clips es necesario tener instalado ROS en una máquina con Linux (ROS: http//ros.org), además se necesita la librería pyCLIPS (http://pyclips.sourcefoge.net), que sirve para embeber al interprete de CLIPS en un programa hecho en el lenguaje de Python; por último se necesitan las librerías ros_pyclips_node y ros_pyclips_service desarrolladas para esta tesis que sirven para crear fácilmente módulos de Python que levanten la comunicación a través de tópicos y servicios entre CLIPS y ROS.

ROS-PYCLIPS consiste básicamente en dos partes: La parte de CLIPS y la parte de Python. La parte de Python es básicamente un módulo que inicializa la conexión con ROS y el intérprete de CLIPS. También define funciones que serían llamadas desde CLIPS, particularmente funciones para enviar y recibir comandos. La parte de CLIPS es un programa que se carga en el interprete de CLIPS al iniciar ROS-PYCLIPS.

ROS-PYCLIPS hace uso de una interfaz desarrollada en el laboratorio de bio-Robótica. Esta interfaz permite controlar algunos parámetros del interprete, como mostrar o no algunas características tales como las reglas o hechos que se van activando y generando, también permite modificar el número de reglas que se ejecutan cada vez que se presiona el botón RUN, permitiendo al usuario ejecutar el plan paso a paso, muy útil para depurar el código, o ejecutar los planes sin pausas.

La figura C-1 muestra la interfaz gráfica de ROS-PYCLIPS

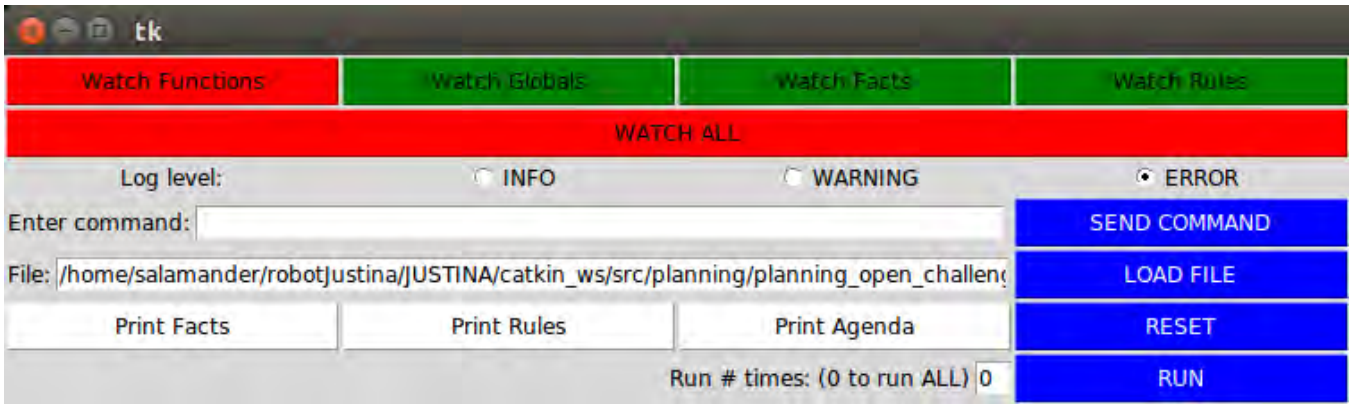


Figura C-1 Interfaz Gráfica de ROS-PYCLIPS.

Los archivos se pueden cargar en al campo File al presionar el botón LOAD FILE, la extensión de los archivos puede ser .clp (extensión de un archivo de CLIPS) o .dat que contienen una lista de archivo .clp que se cargaran conjuntamente. El archivo que se observa en la figura C-1 es una lista de los archivos necesarios para ejecutar la prueba del Open Challenge.

C.1. Funciones de envío y recepción de mensajes de CLIPS

Los archivos de CLIPS que se cargan al inicializar ROS-PYCLIPS incluyen funciones que se pueden utilizar en conjunto con reglas para diferentes programas, dichas funciones son las que permiten el envío y recepción de mensajes así como la limpieza de hechos para no perjudicar la recepción de mensajes. A continuación se presentan las funciones:

sleep: Recibe el tiempo en milisegundos. Esta función evita que Python ejecute al intérprete de CLIPS durante el periodo de tiempo recibido, incluso si se recibe el mensaje. En ese caso se hará el assert del nuevo mensaje recibido pero no se ejecutara ninguna regla hasta que el tiempo de sleep haya terminado.

Send-command. Envía el nombre del comando, id, parámetros del comando y opcionalmente timeout y número de intentos.

Send-response: Recibe el nombre del comando, id, resultado (1 o 0) y parámetros de respuesta.

Ahora con la ayuda de la librería ros_pyclips_node, ROS-PYCLIPS genera los siguientes hechos, correspondientes a distintos eventos:

Cuando ROS envía una respuesta a CLIPS: **(ROS_answer ?cmd ?sym ?result ?params).** La primera y ultima variable son cadenas de texto, la segunda variable corresponde al símbolo del comando y result es 1 o 0.

Cuando CLIPS envía un comando a ROS: **(ROS_cmd ?cmd ?id ?params).** La variable ?cmd es el símbolo del comando ?params es una cadena de texto con los parámetros iniciales de la acción a realizar.

Cuando transcurre el plazo de tiempo de un timer: **(ROS_timer ?sym).** el símbolo corresponde con el símbolo utilizado en el llamado de la función setTIMER.

Una vez que un comando es enviado desde CLIPS hasta ROS la librería ros_pyclips_node publica en los siguientes tópicos de ROS dependiendo del comando que se trate:

Comando para activar la voz del robot:

```
CmdSpeech => rospy.Publisher('/planning_clips/cmd_speech')
```

Comando para ejecutar la interpretación de un comando de voz:

```
CmdInt => rospy.Publisher('/planning_clips/cmd_int')
```

Comando para activar la petición de confirmación del comando:

```
CmdConf => rospy.Publisher('/planning_clips/cmd_conf')
```

Comando para obtener las tareas del Plan:

```
CmdGetTask => rospy.Publisher('/planning_clips/cmd_task')
```

Comando para solicitar que el robot navegue a cierta localización:

```
CmdGoto => rospy.Publisher('/planning_clips/cmd_goto')
```

Comando para enviar la respuesta con el generador de voz del robot a una pregunta del usuario:

```
CmdAnswer = rospy.Publisher('/planning_clips/cmd_answer')
```

Comando para iniciar la detección de objetos o personas:

```
CmdFindObject = rospy.Publisher('/planning_clips/cmd_find_object')
```

Comando para realizar un movimiento con el brazo del robot:

```
CmdMoveActuator = rospy.Publisher('/planning_clips/cmd_move_actuator')
```

Comando para soltar un objeto:

```
Drop = rospy.Publisher('/planning_clips/cmd_drop')
```

Comando para tomar un objeto:

```
Grasp = rospy.Publisher('/planning_clips/cmd_grasp')
```

APÉNDICE D

COMANDOS ALEATORIOS

En este apéndice se describirán los planes creados para los comandos aleatorios generados mas significativos, resultaría muy difícil y extenso poner todos los comandos que se pueden generar por lo que se desecharon comandos que resultaron ser repetitivos en cuanto las acciones que se deben de llevar a cabo para completarlos.

Cada plan generado tiene un número finito de tareas a realizar y para cada tarea existe un identificador “action_type” que posteriormente se utilizara para dividir la tarea en subtareas, para más información de los “action_type” consulte el tema D.1 Acciones en este mismo apéndice. A continuación se describen los planes generados para los siguientes comandos.

Comando Interpretado:

find a person in the kitchen and say your name

```
(task (plan user_speech)
      (action_type find_person_in_room)
      (params kitchen)
      (step 1))
(task (plan user_speech)
      (action_type wait_for_user_instruction)
      (params your_name)
      (step 2))
```

Comando Interpretado:

find a person in the bathroom and answer a question

```
(task (plan user_speech)
          (action_type find_person_in_room)
          (params bathroom) (step 1))
(task (plan user_speech)
          (action_type wait_for_user_instruction)
          (params a_question) (step 2))
```

Comando Interpretado:

find a person in the kitchen and say the date

```
(task (plan user_speech)
     (action_type find_person_in_room)
     (params kitchen)
     (step 1))
(task (plan user_speech)
     (action_type wait_for_user_instruction)
     (params )
(step 2))
```

Comando Interpretado:

find a person in the bedroom and say what day is tomorrow

```
(task (plan user_speech)
     (action_type find_person_in_room)
     (params bedroom)
     (step 1))
```

```
(task (plan user_speech)
      (action_type wait_for_user_instruction)
      (params what_day_is_tomorrow)
(step 2))
```

Comando Interpretado:

find a person in the office and tell what time is it

```
(task (plan user_speech)
      (action_type find_person_in_room)
      (params office)
      (step 1))
(task (plan user_speech)
      (action_type wait_for_user_instruction)
      (params what_time_is_it)
      (step 2))
```

Comando Interpretado:

get into the office and look for the soup

```
(task (plan user_speech)
      (action_type update_object_location)
      (params soup office)
      (step 1))
(task (plan user_speech)
      (action_type get_object)
      (params soup office)
      (step 2))
```

Comando Interpretado:

get into the bedroom and find pringles

```
(task (plan user_speech)
      (action_type update_object_location)
      (params pringles bedroom)
      (step 1))
(task (plan user_speech)
      (action_type get_object)
      (params pringles bedroom)
      (step 2))
```

Comando Interpretado:

go to the kitchen and look for the coke

```
(task (plan user_speech)
      (action_type update_object_location)
      (params coke kitchen)
      (step 1))
(task (plan user_speech)
      (action_type get_object)
      (params coke kitchen)
      (step 2))
```

Comando Interpretado:

reach the bedroom and look for sponge

```
(task (plan user_speech)
      (action_type update_object_location)
      (params sponge bedroom)
      (step 1))
(task (plan user_speech)
      (action_type get_object)
      (params sponge bedroom)
      (step 2))
```

Comando Interpretado:

navigate to the office and find the soup

```
(task (plan user_speech)
      (action_type update_object_location)
      (params soup office)
      (step 1))
(task (plan user_speech)
      (action_type get_object)
      (params soup office)
      (step 2))
```

Comando Interpretado:

get the coke from the kitchen table and deliver it to anna in the kitchen

```
(task (plan user_speech)
      (action_type update_object_location)
      (params coke kitchen_table )
      (step 1))
(task (plan user_speech)
      (action_type get_object)
      (params coke kitchen_table)
      (step 2))
(task (plan user_speech)
      (action_type find_person_in_room)
      (params anna kitchen) (step 3))
(task (plan user_speech)
      (action_type handover_object)
      (params coke)
      (step 4))
```

Comando Interpretado:

take the soup form the desk and bring it to the desk

```
(task (plan user_speech)
      (action_type update_object_location)
      (params soup desk )
      (step 1))
(task (plan user_speech)
```

```
      (action_type get_object)
      (params soup desk)
      (step 2))
(task (plan user_speech)
      (action_type save_position)
      (params current_loc)
      (step 3))
(task (plan user_speech)
      (action_type deliver_in_position)
      (params soup current_loc)
      (step 4))
```

Comando Interpretado:

grasp the coke from the desk and take it to the desk

```
(task (plan user_speech)
      (action_type update_object_location)
      (params coke desk )
      (step 1))
(task (plan user_speech)
      (action_type get_object)
      (params coke desk)
      (step 2))
(task (plan user_speech)
      (action_type save_position)
      (params current_loc)
      (step 3))
```

```
(task (plan user_speech)
      (action_type deliver_in_position)
      (params coke current_loc)
      (step 4))
```

Comando Interpretado:

grasp the soup from the desk and deliver it to alfred at the office

```
(task (plan user_speech)
      (action_type update_object_location)
      (params soup desk )
      (step 1))
(task (plan user_speech)
      (action_type get_object)
      (params soup desk)
      (step 2))
(task (plan user_speech)
      (action_type find_person_in_room)
      (params alfred office)
      (step 3))
(task (plan user_speech)
      (action_type handover_object)
      (params soup)
      (step 4))
```

Comando Interpretado:

grasp the coke from the desk and take it to me

```
(task (plan user_speech)
      (action_type update_object_location)
      (params coke desk )
      (step 1))
(task (plan user_speech)
      (action_type get_object)
      (params coke desk)
      (step 2))
(task (plan user_speech)
      (action_type save_position)
      (params current_loc)
      (step 3))
(task (plan user_speech)
      (action_type deliver_in_position)
      (params coke current_loc) (step 4))
```

Comando Interpretado:

find a person in the kitchen and follow her to the bathroom

```
(task (plan user_speech)
      (action_type update_object_location)
      (params man kitchen )
      (step 1))
(task (plan user_speech)
```

```
      (action_type get_object)
      (params man bathroom)
      (step 2))
```

Comando Interpretado:

get into the kitchen, look for a person and answer a question

```
(task (plan user_speech)
      (action_type find_person_in_room)
      (params kitchen)
      (step 1))
(task (plan user_speech)
      (action_type wait_for_user_instruction)
      (params a_question)
      (step 2))
```

Comando Interpretado:

navigate to the bedroom, find a person and follow her to the kitchen

```
(task (plan user_speech)
      (action_type update_object_location)
      (params man bedroom )
      (step 1))
(task (plan user_speech)
      (action_type get_object)
      (params man kitchen)
      (step 2))
```

Comando Interpretado:

find a person in the kitchen and follow her

```
(task (plan user_speech)
     (action_type update_object_location)
     (params man kitchen ) (step 1))
(task (plan user_speech)
     (action_type get_object)
     (params man )
     (step 2))
```

Comando Interpretado:

find the calling person in the office and follow her to the kitchen

```
(task (plan user_speech)
     (action_type update_object_location)
     (params man_call office )
     (step 1))
(task (plan user_speech)
     (action_type get_object)
     (params man_call kitchen)
     (step 2))
```

D.1 Acciones

Como se explico a lo largo de la tesis en cada plan una tarea consta de un identificador “action_type” que posteriormente se usara para formar las subtareas. Cabe mencionar que la división en subtareas no solo depende del “action_type” si no que también depende del número de parámetros que tenga cada tarea. A continuación se describen todos los “action_type” que pueden aparecer en un plan y su división en subtareas.

Action_type: find_person_in_room (Busca persona en una habitación)

Parametros 1: ?place

```
(plan (name ?name) (number 1)(actions go_to_place ?place)
(plan (name ?name) (number 2)(actions find-object person)
(plan (name ?name) (number 3)(actions go_to_person person)
```

Action_type: find_person_in_room (Busca una persona específica en una habitación)

Parametros 2: ?place ?namePerson

```
(plan (name ?name) (number 1)(actions go_to_place ?place)
(plan (name ?name) (number 2)(actions find-object ?namePerson)
```

Action_type: wait_for_user_instruction (Espera por la instrucción del usuario)

Parametros 1: ?instruction

```
(plan (name ?name) (number 1)(actions user_instruction ?instruction)
```

Action_type: update_object_location (Navega hasta la localización del objeto)

Parametros 2: ?object ?place

```
(plan (name ?name) (number 1)(actions go_to_place ?place)
```

Action_type: get_object (Busca y toma un objeto ubicado en cierta localización con los manipuladores del robot)

Parametros 1: ?object

```
(plan (name ?name) (number 1)(actions ask_for ?object)
(plan (name ?name) (number 2)(actions go_to ?object)
(plan (name ?name) (number 3)(actions attend ?object)
(plan (name ?name) (number 4)(actions find-object ?object)
(plan (name ?name) (number 5)(actions move manipulator ?param)
(plan (name ?name) (number 6)(actions grab manipulator ?param)
```

Action_type: get_object (Busca una persona, cuando la encuentra la comienza a seguir hasta cierto lugar)

Parametros 2: ?man ?place

```
(plan (name ?name) (number 1)(actions find-object-man ?man ?place)
```

Action_type: put_object_in_location (Deja un objeto en cierta localización con los manipuladores del robot)

Parametros 2: ?placement ?object

```
(plan (name ?name) (number 1)(actions go_to_placement ?placement)
(plan (name ?name) (number 2)(actions attend ?object)
(plan (name ?name) (number 3)(actions move manipulator ?object)
(plan (name ?name) (number 4)(actions drop manipulator ?object)
```

Action_type: save_position (Recupera o recuerda la localización donde al robot le hicieron una petición y navega hasta ese lugar)

Parametros 1: ?current_loc

```
(plan (name ?name) (number 1)(actions go_to_place ?current_loc)
```

Action_type: deliver_in_position (Deja un objeto en la localización donde se hizo la petición con los manipuladores del robot)

Parametros 2: ?object ?current_loc

```
(plan (name ?name) (number 1)(actions go_to_placement ?current_loc)
(plan (name ?name) (number 2)(actions attend ?object)
(plan (name ?name) (number 3)(actions move manipulator ?object)
(plan (name ?name) (number 4)(actions drop manipulator ?object)
```

Action_type: handover_object (Deja un objeto en las manos de una persona con los manipuladores del robot)

Parametros 1: ?object

```
(plan (name ?name) (number 1)(actions move manipulator person)
(plan (name ?name) (number 2)(actions drop manipulator ?object)
```

REFERENCIAS BIBLIOGRÁFICAS

[1] Richard E. Fikes et al. “STRIPS: A New Approach to the application of Theorem Proving to Problem Solving”, presented at the 2nd IJCA, Imperial College, London, England, september 1-3, 1971.

[2] Nebel, Backstrom, “Complexity results for SAS+ planning”, Computational Intelligence, Volume 11, Number 4, 1995.

[3] Ruoyun Huang et al. “SAS+ Planning as Satisfiability”, Journal of Artificial Intelligence Reasearch 43, 2012.

[4] Helmert, M. (2006). “The Fast Downward planning system”. *Journal of Artificial Intelligence Re- search*, *26*, 191–246.

[5] Helmert, M., Haslum, P., & Hoffmann, J. (2008). “Explicit-State Abstraction: A New Method for Generating Heuristic Functions”. In *Proceedings of AAAI Conference on Artificial Intelligence*.

[6] Richter, S., Helmert, M., & Westphal, M. (2008). “Landmarks Revisited”. In *Proceedings of AAAI Conference on Artificial Intelligence*.

[7] Chen, Y., Huang, R., & Zhang, W. (2008). “Fast Planning by Search in Domain Transition Graphs”. In *Proceedings of AAAI Conference on Artificial Intelligence*.

[8] Chen, Y., Huang, R., Xing, Z., & Zhang, W. (2009). “Long-distance mutual exclusion for planning”. *Artificial Intelligence*, *173*, 197–412.

[9] Peter E. Hart, Nils J. Nilson, Bertram Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”, IEEE transactions of systems science and cybernetics, vol. Ssc-4, no. 2, july 1968.

[10] Joseph C. Culberson, Jonathan Scheaffer, “Pattern Databases”, Computational Intelligence, volume 14, number 3, 1998.

[11] Stefan Edelkamp, “Planning with Pattern Databases”, In Proc. ECP 2001, pages 13–24.

[12] Malte Helmert, Patrik Haslum, and Jo¨rg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In Proc. ICAPS 2007, pages 176–183, 2007.

[13] Malte Helmert, Carmel Domshlak. “Landmarks, critical paths and abstractions: What’s the difference anyway?” In Proc. ICAPS 2009, pages 162–169, 2009.

[14] Raz Nissim, Jorg Hoffman, Malte Helmert, “Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstraction in Optimal Planning”, International Joint on Artificial Intelligence, 2010.

[15] Kutluhan Erol, James Handler, and Dana S. Nau. “Semantics for Hierarchical Task-Network Planning”. Technical Report CS-TR-3239, UMIACS-TR-94-31, ISR-TR-95-9, University Of Maryland, 1994.

[16] J. Bohren and S. Cousins. “The smach high-level executive [ros news]. Robotics Automation Magazine”, IEEE, 17(4):18-20, Dec 2010.

[17] Luis A. Pineda, Lisset Salinas, Ivan V. Meza, Caleb Rascon, and Gibran Fuentes. Sitlog: A programming language for service robot tasks. International Journal of Advanced Robotic Systems, october 2013.

[18] ROS.org, Getting Started with smach, last edited july 26, 2015, http://wiki.ros.org/smach/Tutorials/Getting%20Started.

[19] Thomas Keller, Patrik Eyerich, “PROST: Probabilistic Planning Base don UCT”, Association for the Advancement of Artificial Intelligence (www.aaai.org), 2012.

[20] Jesus Savage, Adalberto Llarena, Gerardo Carrera, Sergio Cuellar, David Esparza, Yukihiro Minami, and Ulises Peñuelas. Virbot: A system for the operation of mobile robots. In Ubbo Visser, Fernando Ribeiro, Takeshi Ohashi, and Frank Dellaert, editors, RoboCup 2007: Robot Soccer World Cup XI, July 9-10, 2007, Atlanta, GA, USA, volume 5001 of Lecture Notes in Computer Science, pages 512-519. Springer, 2007.

[21] Kutluhan Erol, James Handler, and Dana S. Nau. Semantics for Hierarchical Task-Network Planning. Technical Report CS-TR-3239, UMIACS-TR-94-31, ISR-TR-95-9, University Of Maryland, 1994.

[22] Malik Ghallab, Dana Nau, and Paolo Traverso. Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[23] Vázquez Sánchez Samuel Salvador. “Lenguaje natural y planeación automatizada para robots de servicio doméstico”. Master’s thesis, UNAM, 2015.

[24] Schank, R., "A Conceptual Dependency Representation for a Computer Oriented Semantics", Ph.D. Thesis University of Texas, Austin 1969 (Also available as Stanford AI Memo 83, Stanford Artificial Intelligence Project, Computer Science Department, Stanford University, Stanford, California.)

[25] CLIPS Reference Manual. Volume I, Basic Programming Guide, 2007 .

[26] Revuelta Adrián. “Planeación de acciones utilizando una máquina de inferencias y modelos de redes de petri”. Master’s thesis, UNAM, 2014.

[27] Rina Dechter, Judea Pearl, “Generalized best-first search strategies and the optimality of A*”, Journal of the ACM, volume 32, issue 3, 505--536, 1985.

[28] S. Koenig and M. Likhachev, “D*Lite”, en Artificial Intelligence, Eighteenth National Conference 2002, paginas 476-483.

[29] Wolfe, Jason et al, “Combined Task and Motion Planning for Mobile Manipulation”, Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010).

[30] Jesús Savage et al. “Pumas@Home 2016 Team Description Paper”, UNAM, 2016.