Aprendizaje de Programas Teleo-Reactivos para Robótica Móvil

por

Blanca Alicia Vargas Govea

Tesis sometida como requisito parcial para obtener el grado de

DOCTORA EN CIENCIAS EN LA ESPECIALIDAD DE CIENCIAS COMPUTACIONALES

en el

Instituto Nacional de Astrofísica, Óptica y Electrónica

Octubre 2009

Tonantzintla, Puebla

Supervisada por:

Dr. Eduardo F. Morales Manzanares

Dedicatoria

A mis padres, por estar siempre, por poder contar con éllos, por ser los mejores amigos y consejeros que alguien puede tener.

Agradecimientos

Al Dr. Eduardo Morales quien ha sido mi asesor desde la maestría, por su guía y por compartir su experiencia y conocimientos. Sin embargo, más allá del aspecto académico, su valor humano es lo que más aprecio. Muchos pueden tener conocimiento pero son pocos los que pueden con su ejemplo y actitudes, enseñar a ser mejor persona. Ha sido muy divertido y enriquecedor trabajar con usted. Gracias.

Al Dr. Enrique Sucar por haberme considerado para participar en la ya legendaria Cátedra de Robótica Móvil, de donde surgió esta tesis. Ha sido una gran experiencia. Gracias por sus sugerencias, consejo y disponibilidad a lo largo de estos años.

A los Doctores: Angélica Muñoz, Jesús González, Leopoldo Altamirano y René Mac Kinney por las aclaraciones, los valiosos comentarios, el tiempo y el esfuerzo dedicado a mejorar esta tesis.

Al Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE) por las facilidades y los recursos brindados para el desarrollo de esta tesis. Agradezco al Departamento de Ciencias Computacionales la ayuda proporcionada durante estos años.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo proporcionado a través de la beca número 203873.

A Rita, Carlos y Rafa por los años compartidos en el cubo 8216. Siempre me pregunté cómo Rita podía cubrir tantas facetas de vida. Gracias a Carlos por sus valiosas opiniones y consejos. Y a Rafa, quien ha sido un gran apoyo, compañía incondicional y de quien he aprendido mucho.

A los miembros del laboratorio de robótica, particularmente Elva y Josué con quienes se vivió el surgimiento de Markovito. También a Alberto y Cristina por sus útiles sugerencias para sobrevivir en el periodo de revisiones.

A Maya y Marco Aurelio quienes durante la etapa de escritura hicieron alegre la hora de la comida, el único momento del día en el que salía de la ermitañez y convivía con humanos. Gracias por su amistad.

A Héctor, a quien conozco desde hace como mil años, por haber dado vida a Markovito al implementar su arquitectura inicial, por compartir sus experimentos sobre ademanes y sobre todo, por su amistad y confianza.

A Víctor, quien de forma trasatlántica ha estado siempre presente siendo un querido amigo y un enorme apoyo en este camino, pero sobre todo por compartir sus Ptero-historias de vida y por la invención del mundo Jam. Desde su mundo de fantasía, el Pterodáctilo-desvencijado y el Jam seguramente sonríen.

A mis padres, quienes no sólo han estado presentes durante todo el trayecto de la tesis con su apoyo en todos los aspectos sino también contribuyendo activamente durante el proceso de revisión. Sin su ayuda, diversas aclaraciones solicitadas por los revisores habrían sido difíciles de interpretar.

Y después del trabajo de tesis realizado, de constatar que lo mostrado en la ficción no es más que eso y de ver lo lejos que está el humano de poder dotar de autonomía, de inteligencia a una entidad no viva valoro más la creación, la inteligencia natural que no se alcanza a comprender. Agradezco a esa energía superior que independientemente de cualquier religión hace posible que existamos y compartamos este espacio de tiempo y vida.

Resumen

En sus inicios, los robots desempeñaban tareas repetitivas en ambientes industriales muy controlados. Al paso del tiempo las aplicaciones de los robots se han ido diversificando hasta el punto de empezar a integrarse a la sociedad desempeñando tareas en casas y oficinas. Esta integración implica nuevas dificultades y retos pues las habilidades que se necesitan en ambientes no controlados son mayores. El robot está expuesto a personas en movimiento, mobiliario que cambia de posición por lo que el robot necesita, por ejemplo, habilidades de desplazamiento, evasión de obstáculos, entre otras que le permitan integrarse al ambiente. Dos de las características que un robot debe tener para lograr cierta autonomía son: i) contar con habilidades para realizar tareas y ii) responder adecuadamente a eventos en ambientes dinámicos. La programación de un robot no es fácil y la incorporación de este tipo de habilidades y comportamientos puede ser un proceso largo y tedioso. Estos aspectos han motivado la búsqueda de métodos que permitan que la programación de robots sea más rápida y menos compleja.

En esta tesis se propone una metodología de aprendizaje que simplifica la programación de robots móviles en ambientes interiores. Se utilizan dos técnicas, la primera es el aprendizaje de Programas Teleo-Reactivos (PTRs) [Nilsson, 1994], que son programas que están orientados a metas y al comportamiento en ambientes dinámicos. La segunda técnica, conocida como clonación [Michie y Sammut, 1995] ha sido utilizada en el aprendizaje de habilidades a partir de ejemplos y su objetivo es la inducción de modelos o conjuntos de reglas que puedan integrarse como controladores en un sistema.

Se muestra cómo un robot aprende PTRs a partir de ejemplos o trazas generados por una persona. El usuario guía al robot a ejecutar la tarea y el robot aprende a realizarla. Se propone un aprendizaje en tres fases: (i) transformación de trazas con información de bajo nivel de los sensores a trazas de alto nivel mediante un proceso de identificación de marcas naturales. Esta transformación facilita el uso de algoritmos de aprendizaje, (ii) aprendizaje de PTRs básicos que expresan cuándo ejecutar una acción para realizar tareas simples usando un sistema de Programación Lógica Inductiva (ILP), y (iii) aprendizaje de PTRs jerárquicos que expresan cómo lograr metas siguiendo secuencias particulares de acciones. Se introduce un algoritmo para inducción de gramáticas relacionales que pueden usarse como programas de control y también para clasificar secuencias.

Se presentan experimentos realizados para el aprendizaje de PTRs de navegación para un robot móvil en ambientes dinámicos y diferentes a aquéllos en los que se le entrenó. Los

experimentos fueron realizados en simulación y en un robot real. Los resultados muestran que los PTRs aprendidos permiten controlar al robot con precisión mayor a 80 % en ambientes dinámicos. Se muestran los resultados comparativos en simulación de los PTRs con un enfoque no reactivo. Se aprendieron gramáticas para reconocimiento de nueve ademanes. Las gramáticas aprendidas se probaron para clasificar nuevas secuencias obteniendo resultados competitivos en comparación con un enfoque [Avilés-Arriaga et al., 2006] que utiliza modelos ocultos de Markov (HMM). Las gramáticas aprendidas son expresivas y permiten identificar sub-tareas para el caso de navegación y sub-ademanes. Ésto no es posible con otros enfoques de aprendizaje.

Abstract

In their beginnings, robots performed repetitive tasks in industrial environments or extremely controlled laboratories. At present, the scope of robotic applications has been extended and now robots have reached home and office environments developing service or assistant tasks. Their integration to society involves challenges and increasing difficulties because the robots require skills in non-controlled and populated environments.

An autonomous robot needs skills to perform tasks and to react accordingly to unexpected events in dynamic environments. Programming a robot to have these skills is a complex and time consuming process. These difficulties have motivated to look for methods to ease the programming effort.

In this thesis a learning methodology to simplify the programming effort of mobile robots is proposed. The methodology uses two techniques: (i) Teleo-Reactive Programs (PTRs for its name in Spanish) which have proved to be an effective framework to continuously perform a set of actions to achieve particular goals and to react in the presence of unexpected events, and (ii) behavioural cloning, a technique to learn skills by example inducing a model that can be used as a system controller.

In this thesis, it is shown how a robot can learn PTRs from human guided traces. A user guides a robot to perform a task and the robot learns how to perform such task in similar dynamic environments. Our approach follows three steps: (i) it transforms traces with low-level sensor information into high-level traces based on natural landmarks, (ii) it learns PTRs that express when to perform an action to achieve simple tasks using an Inductive Logic Programming (ILP) system, and (iii) it learns hierarchical PTRs that express how to achieve goals by following particular sequences of actions using FOSeq, a proposed grammar induction algorithm.

We test the approach in a robotics scenario with both simulated and real environments and show that the robot is able to accomplish several navigation tasks with the learned TRPs in different dynamic and unknown environments with good precision. It is also shown a comparison in simulation against a non-reactive approach.

FOSeq was also used to learn gesture grammars with competitive results when compare with a recent state-of-the-art system. Classification accuracy with FOSeq is 97.34 % and with HMM is 97.56 %. Learned grammars are expressive and can be used to recognize and to execute tasks or sub-tasks and gestures or sub-gestures. This is not possible with other learning approaches.

Contenido

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

1 Introducción

1.1 Motivación

El uso de robots se ha extendido más allá del ámbito industrial. La imagen de un robot en una fábrica o en los laboratorios se está desvaneciendo dando paso a una generación de robots que realizan tareas cotidianas en beneficio humano. A este tipo de robots se les llama robots de servicio. Con esta tendencia, tareas que tradicionalmente han sido realizadas por personas ahora están empezando a ser realizadas por robots. Tomar fotografías en un evento social [Byers et al., 2001], asistir a ancianos en un asilo [Pineau et al., 2003] o guiar a turistas en un museo [Thrun et al., 2000] son ejemplos de tareas en cuya solución con robots se está trabajando.

De acuerdo al reporte de la Federación Internacional de Robots [ifr url, 2008] el rubro de los robots de servicio es el que presenta una mayor tasa de crecimiento en comparación con los robots de tipo industrial. Hasta el año 2007 se habían vendido aproximadamente 5.5 millones de robots de servicio y al menos 12.2 millones se proyecta que sean vendidos entre 2008 y 2011. Estos datos muestran la tendencia de la sociedad a incorporar robots a su vida diaria. Este panorama ha originado una diversificación en las tareas a realizar por los robots de servicio por lo que ahora es posible encontrarlos en la agricultura, hotelería, medicina, química, servicio militar, mensajería y para uso doméstico. Sin embargo, para que un robot de servicio se incorpore exitosamente en su entorno debe contar con habilidades para llevar a cabo tareas funcionales y de interacción. Necesita tener mecanismos que le proporcionen autonomía para tomar decisiones.

Tradicionalmente se programa a un robot de forma explícita para realizar tareas. Éste es un proceso difícil y largo por lo que resulta interesante pensar en un robot que sea capaz de

aprender a realizar tareas de servicio de forma simple. Sería útil que se le pudiera "instruir" mostrándole qué hacer y no cómo. De esta manera se reduciría el tiempo y la dificultad de programación, lo que repercutiría positivamente en el desarrollo de un mayor número de tareas en menor tiempo.

Se han propuesto diversas técnicas de aprendizaje como por ejemplo, métodos basados en redes neuronales y algoritmos de aprendizaje como C4.5 [Quinlan, 1993] que son buenos creando políticas simples a partir de ejemplos. Una aplicación basada en redes neuronales es el sistema ALVINN [Pomerleau, 1991]. ALVINN observa las acciones de un conductor y usa una red neuronal de retro-propagación para generalizar imágenes del camino y su correspondiente dirección como salida. ALVINN aprendió a guiar un vehículo por caminos pavimentados y no pavimentados de un solo carril, y múltiples carriles marcados y no marcados.

El aprendizaje puede realizarse también por métodos no supervisados usando métodos de aprendizaje por refuerzo como Q-learning [Sutton y Barto, 1998] en donde la opción más simple es construir una tabla de pares estado-acción.

Los problemas de estos enfoques son la inflexibilidad de manejar las distintas metas que un agente puede tener. La política aprendida para una tarea puede ser inútil para otra. Estos enfoques puede generar un espacio de estados tan grande que resulte intratable.

La introducción de operadores STRIPS [Fikes y Nilsson, 1971] permitieron a planificadores no considerar todos los atributos para representar un estado del mundo. Sin embargo, el aprendizaje de modelos simbólicos de acciones para planificación puede ser apropiado para ciertos dominios pero no es aplicable a dominios con variables continuas y sensores susceptibles al ruido. Recientemente el uso de enfoques relacionales ha tomado importancia por sus ventajas sobre otro tipo de técnicas, como por ejemplo: nivel de expresividad alto, facilita el aprendizaje de descripciones de estados y modelos de acciones, permite incorporar posible conocimiento del dominio a diferencia de algoritmos proposicionales, facilita la transferencia a dominios parecidos.

En robótica, un aspecto muy importante en los paradigmas de control clásicos es la representación del mundo. Mientras que un algoritmo deliberativo genera planes basándose en la representación del mundo, un enfoque reactivo es necesita tener conocimiento del mundo, basta con tener lecturas de los sensores del estado actual del robot para decidir la acción a ejecutar. En esta tesis se seleccionó el formalismo teleo-reactivo [Nilsson, 1994] que es considerado un control de nivel-medio puesto que los Programas Teleo-Reactivos necesitan una representación del mundo pero únicamente necesitan el estado actual del robot para ejecutar las acciones. En el capítulo se describe el formalismo teleo-reactivo. El uso

de representaciones relacionales facilitan el uso de algoritmos para el aprendizaje de las descripciones del mundo. En esta tesis nos interesa generar a partir de información de bajo nivel las descripciones del mundo y las reglas de control para un robot móvil.

Algunos de los problemas comunes en los trabajos de aprendizaje computacional en robótica son los tiempos de aprendizaje largos y resultados poco entendibles al usuario o programador. El propósito de esta tesis es desarrollar una plataforma de aprendizaje que facilite la programación de robots móviles, obteniendo como resultados conjuntos de reglas o predicados de control confiables. El enfoque de solución que se presenta aborda los problemas de: facilidad de programación, robustez en el control obtenido y entendimiento en la representación. A partir de ejemplos: (i) se aprenden automáticamente conjuntos de Programas Teleo-Reactivos [Nilsson, 1994] básicos para robots móviles, (los PTRs son programas lógicos que controlan a un robot en ambientes dinámicos) y (ii) se aprenden gramáticas a partir de secuencias de ademanes para su clasificación.

En esta tesis se busca extraer conocimiento estructurado a partir de conjuntos de cientos de datos que no tienen significado aparente. Este aspecto es muy importante porque una representación estructurada puede proporcionar: (1) facilidad de interpretación por parte de un usuario que no sea experto en el dominio de aplicación, (2) facilidad para usar algoritmos pues en vez de cientos de datos se extraen únicamente los datos significativos, y (3) facilidad para el programador pues es posible generar automáticamente programas que manualmente pueden requerir experiencia en el dominio de aplicación y consumir mucho tiempo para desarrollarlos.

1.2 Problemática a resolver

Para que un robot móvil sea capaz de realizar tareas debe decidir qué acción tomar en cada instante de tiempo. Debe contar con las descripciones del ambiente sensado periódicamente aplicando un conjunto de acciones que puede efectuar. De acuerdo a lo que sensa y a la meta que desea alcanzar, el robot debe tomar una decisión sobre la acción a ejecutar. Éste es un proceso difícil que presenta los siguientes problemas.

Espacio de estados y variables continuas. El espacio de estados es el conjunto de todos los estados que pueden alcanzarse desde un estado inicial por una secuencia de acciones y describe todas las posibles configuraciones del ambiente [Russell y Norvig, 2003]. En robótica, el espacio de estados es continuo, ésto significa que las variables involucradas están definidas por valores reales por lo que es difícil aplicar algoritmos estándar debido

a la información abundante y de bajo nivel que se adquiere con el sensado continuo. Se requiere aplicar tratamientos especiales a los datos adquiridos para que pueda obtenerse un buen resultado de los algoritmos de aprendizaje que se utilicen. Para reducir este problema, estamos utilizando un proceso de identificación de marcas naturales del ambiente junto con una representación relacional que reduce el conjunto de datos de entrada a un conjunto compacto de atributos significativos.

Ambiente dinámico. En un ambiente de oficinas existen eventos dinámicos frente a los que un robot de servicio debe ser capaz de reaccionar correctamente. Si el robot está cumpliendo una meta y ocurre un suceso inesperado debe tener la habilidad de no perder su objetivo principal y cumplirlo. Para lograrlo, el proceso de aprendizaje que presentamos utiliza un enfoque teleo-reactivo en el cual, los programas que se aprenden están orientados a resolver metas y pueden reaccionar ante eventos inesperados.

Tareas, acciones básicas y jerarquías. En términos generales, una tarea es un conjunto de acciones para lograr una meta. Resolver una tarea generalmente involucra resolver sub-tareas que a su vez pueden descomponerse en nuevas sub-tareas hasta llegar a realizar acciones básicas que son indivisibles. Esta descomposición da lugar a una estructura jerárquica cuya raíz es la tarea principal y cuyas hojas son las acciones básicas. Se identifican así dos problemas principales: i) cómo aprender acciones básicas y ii) cómo aprender jerarquías de acciones. Se desarrolló una metodología de aprendizaje para PTRs básicos mediante una técnica conocida como clonación y programación lógica inductiva (ILP). Para el aprendizaje de PTRs jerárquicos se presenta un algoritmo de aprendizaje a partir de secuencias.

Dificultad de programación. La programación de un robot se realiza tradicionalmente a bajo nivel por lo que es un proceso arduo que consume mucho tiempo. Si el robot se programa para realizar una tarea en un lugar específico, al ser cambiado de lugar normalmente tiene que ser reprogramado aunque las características del lugar sean similares. Para resolver este problema utilizamos una representación relacional que permite que al cambiar de ambiente no sea necesario realizar un nuevo proceso de aprendizaje.

Una plataforma de aprendizaje de tareas que simplifique la programación, que integre mecanismos para reducir el espacio de estados y que pueda ser transferible a otros dominios con características similares, permitiría programar robots en menor tiempo. En esta tesis se presenta un esquema de aprendizaje que habilita a un robot móvil a realizar tareas en un ambiente dinámico y reduce los problemas mencionados.

1.3 Objetivos

El principal objetivo de esta tesis es desarrollar una plataforma de aprendizaje relacional de tareas para un robot móvil. A partir de trazas obtenidas mientras el usuario guía a un robot con una palanca controladora (*joystick*), el robot aprende habilidades para realizar las tareas.

En el proceso de aprendizaje se identifican los siguientes objetivos específicos:

- Desarrollar mecanismos que permitan manejar los cientos de datos de bajo nivel recibidos de los sensores en cada instante para construir una representación relacional a partir de información significativa.
- Aprender habilidades a partir de una representación relacional usando trazas generadas por una persona.
- Desarrollar un algoritmo cuya salida sean programas lógicos que expresen la estructura jerárquica de secuencias de acciones.

1.4 Contribuciones

Las principales contribuciones de esta tesis son:

- Representación de alto nivel mediante una transformación de información de bajo nivel (sensores). Para lograr este objetivo se utiliza un proceso de identificación de marcas naturales del ambiente a partir de la información de los sensores del robot. Esta información junto con conocimiento del dominio adicional se usa para generar una representación relacional que facilita el uso de algoritmos de aprendizaje.
- Algoritmo para aprendizaje de conceptos y PTRs básicos usando clonación y programación lógica inductiva. A partir de ejemplos generados por el usuario se aprenden conceptos que identifican zonas del ambiente. Los PTRs que se aprenden utilizan ésos conceptos como conocimiento del dominio, de esta manera, si el robot está por ejemplo en una zona identificada como "libre", el PTR puede utilizar este concepto para avanzar sin peligro. Para aprender PTRs se usan trazas que se obtienen al guiar al robot, se transforman a una representación de alto nivel y se dan como entrada junto con posible conocimiento del dominio (e.g. los conceptos aprendidos) a un sistema de *ILP*.

El resultado es un conjunto de PTRs básicos que permiten al robot realizar la tarea mostrada por el usuario.

- Algoritmo para aprendizaje de PTRs compuestos. Un PTR compuesto está formado por dos o más PTRs básicos. La idea es identificar en las trazas obtenidas al realizar una tarea con PTRs previamente aprendidos con los cuales se puedan construir automáticamente PTRs compuestos. Se presenta un algoritmo que a partir de secuencias aprende programas lógicos que representan la estructura jerárquica de la secuencia de entrada, y se usaron: (i) como programas lógicos en la forma de PTRs compuestos y (ii) como clasificadores de secuencias de ademanes.

1.5 Organización del documento

La tesis está organizada como sigue. El capítulo 2 muestra un panorama del aprendizaje en robótica y se describen los fundamentos de las técnicas utilizadas a lo largo de la tesis.

El capítulo 3 presenta una revisión del estado del arte bajo la perspectiva de las distintas áreas que se abordan en la tesis como son robótica, programas teleo-reactivos (PTRs), aprendizaje relacional y aprendizaje a partir de secuencias.

En el capítulo 4 se describe el aprendizaje de PTRs básicos usando programación lógica inductiva (ILP). El capítulo 5 presenta un algoritmo para el aprendizaje de gramáticas a partir de secuencias. Las gramáticas se usaron para el aprendizaje de PTRs compuestos y para clasificación de ademanes.

El capítulo 6 muestra los resultados experimentales en el robot, tanto en simulación como en un robot real y se presentan los experimentos realizados para clasificación de ademanes. Finalmente, en el capítulo 7 se presentan las conclusiones y las posibles direcciones en las que el trabajo presentado puede mejorarse y extenderse.

2

Conceptos Generales

En este capítulo se presenta un panorama general del aprendizaje en robótica con el fin de ubicar la tesis en esta área. También se describen las técnicas y la terminología que se utilizará a lo largo de la tesis. Los temas que se describen son: (i) clonación de comportamiento, (ii) Programas Teleo-Reactivos (PTRs), (iii) aprendizaje relacional, y (iv) gramáticas de cláusulas definidas (GCDs).

2.1 Introducción

Imaginemos que una mañana cualquiera, una familia va de compras a su almacén favorito y se dirige a la sección de "robots de servicio" a adquirir uno nuevo. El robot que poseen se ha tornado obsoleto y ya no cubre sus necesidades. Eligen el último modelo, que incluye tareas de limpieza, de mensajería, de juegos y de atención a enfermos y mascotas. Llegando a casa desempacan el robot al que configuran de forma muy sencilla, tal como se hace con las televisiones y otros aparatos electrónicos y de inmediato el robot empieza a cumplir con sus funciones. La tendencia del mercado es que este escenario llegue a ser una realidad. Aunque ya existen a la disposición del público productos como pequeños robots para limpieza de pisos [iRobot url, 2008] y también software [Skilligent-url, 2008] para enseñar a robots a realizar tareas, el desarrollo de robots de servicio es un área de oportunidad que está en creciente cambio. En esta tesis se presenta una metodología y aplicación de aprendizaje para tareas de navegación y para reconocimiento de ademanes. Su contribución es un paso más para alcanzar el escenario descrito al inicio de esta sección.

Pero, ¿cómo lograr que una persona que desconoce aspectos técnicos y de programación pueda "instruir" a un robot para realizar tareas? Para tal fin, es necesario que el robot tenga

incorporados mecanismos que le permitan aprender del ambiente. Es en este aspecto donde las técnicas de aprendizaje computacional se han aplicado con éxito. El aprendizaje computacional es un subcampo de la inteligencia artificial que se relaciona con el diseño y desarrollo de algoritmos que permitan a las computadoras "aprender". Se enfoca en extraer información a partir de datos de forma automática usando métodos estadísticos o computacionales. Pero, ¿qué significa que un robot "aprenda"?, ¿cómo se define una tarea de aprendizaje?, ¿cómo se representa? Las respuestas a estas preguntas se abordan en este capítulo y se presentan también las técnicas de aprendizaje usadas a lo largo de la tesis.

2.2 Aprendizaje en robótica

En sus inicios, los robots se utilizaban en fábricas para realizar tareas repetitivas. El entorno era generalmente el mismo y las condiciones extremadamente controladas. En este tipo de ambientes, difícilmente existía el riesgo de que al robot se le presentaran situaciones nuevas y en caso de que ocurriera algún evento inesperado, el robot fallaba pues la secuencia de pasos se interrumpía. A diferencia de este panorama, los robots de servicio tienen que realizar sus tareas en ambientes dinámicos, en el mismo lugar donde las personas viven o trabajan, en presencia de obstáculos fijos como paredes, y obstáculos móviles como muebles y personas. La programación de un robot en este entorno se complica pues es muy difícil predecir todas las situaciones en las que pueda encontrarse el robot. Los métodos de aprendizaje computacional son convenientes en ambientes dinámicos pues las aplicaciones son complejas para ser diseñadas y desarrolladas manualmente [Mitchell, 2006].

Un robot que va a realizar tareas en ambientes reales y dinámicos necesita representar el mundo físico para tomar decisiones con base en esa información. Para establecer la metodología que se seguirá para aprendizaje hay distintos aspectos que se tienen que considerar [Sim et al., 2003]: (i) la representación del ambiente, (ii) lo que el robot va a aprender, y (iii) los métodos de aprendizaje que se aplicarán. A continuación se describen estos aspectos.

Representación del ambiente. La representación del ambiente depende de las capacidades de sensado del robot y del tipo de ambiente en donde se trabaje.

- **Sensores**. Los sensores de los que dispone el robot son determinantes para definir lo que se va a aprender. Las lecturas de los sensores proporcionan la información para construir la representación del ambiente. Así, si disponemos de una cámara, se puede pensar en obtener marcas visuales. Un láser nos dará mediciones de distancia muy

precisas hacia los objetos aunque tiene la limitación de no percibir ciertas superficies como el cristal. Cada sensor tiene distintas limitaciones y alcances que hay que tomar en cuenta para lograr la representación más útil a partir de su información. En esta tesis se utiliza un sensor láser a cuyas lecturas se les aplica un procedimiento de identificación de marcas naturales [Hernández y Morales, 2006] y se usa también un anillo de sonares para detectar objetos en la parte trasera del robot (ver Figura 2.1).

Figura 2.1: Robot Peoplebot de ActivMedia con sensor láser y anillo de sonares

- **Tipo de ambiente**. De acuerdo al tipo de ambiente en el que pueden operar, los robots pueden ser acuáticos, aéreos o terrestres. En esta tesis utilizamos un robot que se desplaza en un medio terrestre en ambientes interiores tipo casa u oficina.

- **Nivel de representación**. La representación del ambiente puede hacerse en dos niveles: (i) bajo nivel que consiste en las lecturas de los sensores de forma cruda y (ii) alto nivel que consiste en procesar los datos de bajo nivel a distintos niveles de abstracción como por ejemplo, transformarlos a aun conjunto de atributos que representen marcas del ambiente y posteriormente a predicados. En esta tesis se usa una representación de alto nivel en lógica de primer orden para describir el ambiente.

¿ Qué va a aprender el robot?

- A partir de la representación del ambiente se puede aprender a identificar, por ejemplo, zonas específicas como pasillos, esquinas, habitaciones, trampas, el mapa completo del ambiente, objetos, personas, etc.

- Políticas de acciones para lograr una meta.

- Comportamientos o habilidades para realizar tareas: navegar, manipular, bailar, limpiar, guiar, etc.

La claridad del conocimiento aprendido también es un aspecto a considerar. Es más sencillo comprender desde el punto de vista de un usuario sin conocimiento de experto lo que el robot está haciendo si se presenta en un lenguaje expresivo.

Dos de los términos más comunes que se encuentran en la literatura sobre aprendizaje en robótica son "habilidad" y "tarea". Se dice que los robots necesitan "habilidades" para realizar "tareas" en determinados contextos. A continuación se definen los conceptos y la relación entre ellos consideradas en esta tesis.

Habilidad. De acuerdo a [Kaiser y Dillmann, 1996], una habilidad es la capacidad aprendida para realizar algo de forma competente. A una habilidad se le llama a menudo comportamiento [Konidaris, 2008]. Si se considera que un robot es el que aprenderá una habilidad, significa que para un estado dado, el robot debe ejecutar una acción con el fin de lograr una meta asociada a una habilidad particular [Kaiser y Dillmann, 1996]. La acción ejecutada debe ser el resultado de una decisión competente relacionada con la meta a alcanzar. En el contexto de "clonación de comportamiento" [Michie y Sammut, 1995] una habilidad es la capacidad para realizar una actividad que no puede ser descrita explícitamente, es tácita en el sentido de que quien posee esa habilidad no se percata de su forma de operación. Por ejemplo, explicar cómo conducir una bicicleta es difícil pues la persona no piensa en variables o atributos que puedan intervenir en el proceso: "giraré el manubrio 30 $grados$ y luego avanzaré 30 $metros$ a una velocidad de 1 $km/hora$", simplemente lo hace, no es su intención hacer un modelo de su habilidad. Tomando en cuenta las ideas anteriores, en esta tesis se considera que una habilidad es la capacidad aprendida para realizar alguna actividad que es difícil de describir de forma explícita. El uso de técnicas de aprendizaje computacional puede facilitar la extracción del conocimiento necesario para inducir controladores que proporcionen a un robot habilidades para realizar tareas. Un robot puede por ejemplo, tener habilidades para: evasión de obstáculos, desplazamiento en un ambiente para llegar a un determinado punto, reconocimiento de personas, sujeción de objetos, seguimiento de un objeto o de una persona.

Tarea. A menudo se usa el término "tarea" de forma indistinta con actividad o proceso. Una tarea se define como el conjunto de pasos o actividades que se necesitan para lograr una meta. En robótica, la descripción de una tarea [Vijaykumar et al., 1987] consiste en: (i) una meta, (ii) una estrategia para lograr la meta y (iii) las condiciones de aplicabilidad que especifiquen si la estrategia de control sigue siendo apropiada para lograr la meta. Para saber

si una condición de aplicabilidad continúa siendo verdadera durante la ejecución de la tarea se necesita un monitoreo continuo. Por ejemplo, si durante la tarea de insertar un perno, la pinza de un robot lo suelta o lo pierde, la tarea de inserción ya no puede realizarse, la condición para que la tarea se realice es que la pinza tenga un perno por lo que la condición ya no se cumple. Y para realizar una tarea, un robot necesita habilidades.

Una tarea puede tener sub-tareas, por ejemplo, si la tarea del robot es navegar de un punto origen a un punto destino, puede tener como sub-tareas: evadir obstáculos orientarse. Navegación, a su vez, puede ser una sub-tarea para una tarea de mayor nivel como por ejemplo, entrega de mensajes a una persona. En el nivel más bajo se encuentran las acciones atómicas [Mataric, 1998] como `avanzar`, `girar a la izquierda`, `girar a la derecha`, `detenerse`. En la Figura 2.2 la Tarea 1 se realiza mediante la ejecución de la sub-tarea 1 y la sub-tarea 2. La sub-tarea 1 necesita ejecutar las acciones atómicas 1 y 2 mientras que la sub-tarea 2 utiliza las acciones atmicas 2 y 3. La Tarea 1 a su vez, podría convertirse en una sub-tarea para una tarea de mayor nivel.

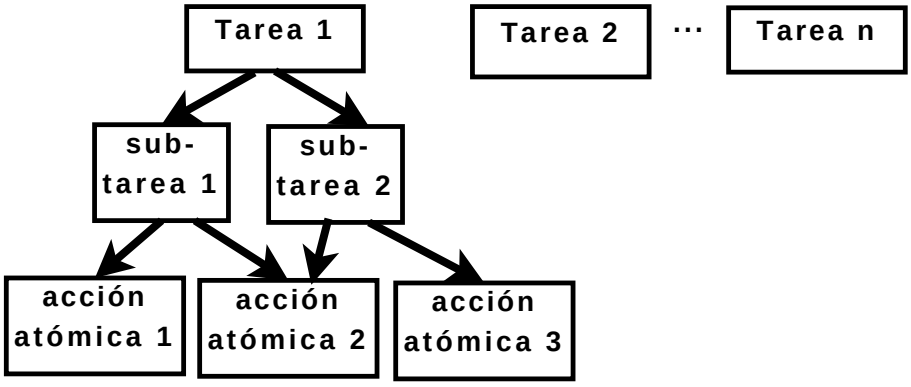


Figura 2.2: Tareas, sub-tareas,acciones atómicas

Métodos de aprendizaje. Los métodos de aprendizaje se pueden clasificar en cuanto al tipo de tarea que abordan y la representación utilizada de los modelos inducidos.

En cuanto al tipo de tarea que abordan dichos métodos pueden ser:

- **Supervisados**. Son métodos cuyo objetivo es aprender una función a partir de ejemplos de entrenamiento. Los ejemplos son pares de entrada (generalmente vectores) y salida. La salida de la función puede ser un valor continuo o puede predecir la clase del ejemplo de entrada. La tarea es predecir el valor de la función para cada ejemplo válido después de ver un número de ejemplos de entrenamiento. Para lograrlo, se tiene

que generalizar para poder reconocer ejemplos nuevos. Ejemplos de estos métodos son las redes neuronales y los árboles de decisión.

- **No supervisados**. En el aprendizaje no supervisado se busca determinar cómo se organizan los datos de entrada puesto que los ejemplos no están etiquetados con una clase. Un ejemplo de estos métodos es el *clustering* que consiste en asignar a cada ejemplo un grupo (*cluster*) de tal forma que los ejemplos del mismo *cluster* sean muy parecidos entre sí, y diferentes de los ejemplos de otros *clusters*.

- **Aprendizaje por refuerzo**. Se refiere a cómo un agente debe seleccionar acciones en un ambiente, maximizando una recompensa a largo plazo. El objetivo de los algoritmos de aprendizaje por refuerzo es encontrar una política que mapee los estados del mundo a las acciones que el agente debe tomar en ésos estados [Sutton y Barto, 1998]. El ambiente se formula generalmente como un proceso de decisión de Markov (MDP) y los algoritmos de aprendizaje por refuerzo en este contexto se relacionan con técnicas de programación dinámica. En estos métodos de aprendizaje no se dan ejemplos de entrenamiento. Algunos ejemplos de estos algoritmos son Q-learning y Sarsa [Sutton y Barto, 1998].

Los métodos de aprendizaje pueden clasificarse también por la representación de los modelos en:

- **Simbólicos**. Usan representaciones simbólicas como reglas de producción o árboles de decisión para describir una clase o clases de ejemplos.

- **Aproximación de funciones**. Aprenden funciones para generalizar y producir salidas para ejemplos no conocidos.

El aprendizaje por imitación es otro método que ha sido aplicado en robótica. El término aprendizaje por imitación es encontrado con frecuencia en la literatura sobre robótica [Billard et al., 2008], y usado de forma indistinta con el término *clonación de comportamiento*. Se observa que los trabajos sobre imitación no hacen referencia alguna a *clonación* y viceversa, a pesar de estar estrechamente relacionados. En esta tesis utilizamos el término *clonación de comportamiento* por lo que se considera necesario aclarar las diferencias entre ambas técnicas.

2.3 Aprendizaje por imitación

El aprendizaje por imitación, también conocido como programación por demostración [Billard et al., 2008] surgió a principios de los ochentas como camino para automatizar la programación manual de robots industriales. Las demostraciones se hacían por tele-operación, es decir, guíando al robot a distancia y las secuencias estado-acción se registraban. La información de los sensores era segmentada en sub-metas discretas que correspondían a puntos a lo largo de la trayectoria. Las acciones básicas o primitivas eran generalmente movimientos punto-punto para alcanzar las submetas. Así, la tarea demostrada era segmentada en una secuencia de transiciones estado-acción-estado. Esta secuencia era convertida en reglas *if-then* que describían los estados y las acciones de acuerdo a relaciones simbólicas, tales como "cerca-de", "mover-hacia", "mover-sobre". A estos símbolos se le asociaban rangos numéricos que se daban como conocimiento *a priori* al sistema. Una demostración completa se representaba en un grafo donde cada nodo representaba un estado y los arcos representaban las acciones.

Posteriormente se sugirió el uso de técnicas de aprendizaje computacional y surgieron así preguntas como: ¿cómo generalizar una tarea?, ¿cómo reproducir una habilidad en situaciones desconocidas?, ¿cómo evaluar un intento?, y ¿cómo definir el papel del usuario durante el aprendizaje?

En los trabajos iniciales se hizo notar la importancia de proporcionar un conjunto de ejemplos numeroso y diverso. Se incorporó al usuario como parte activa del aprendizaje al proporcionarle ejemplos que no hubieran sido cubiertos. Actualmente el trabajo que se realiza es muy similar a los iniciales. Se ha avanzado en la forma de guiar al robot, la forma de interacción y el tipo de sensores. Se incorporaron técnicas para abordar aspectos como la generalización entre demostraciones y la generalización del movimiento en distintas situaciones. Ejemplos de las técnicas son: redes neuronales, lógica difusa y modelos ocultos de Markov. La tendencia en el aprendizaje por imitación es basarse en el comportamiento animal y de infantes. La "bio-inspiración", como se le llama en el área tiene como uno de sus principales componentes la representación sensorimotora. Actualmente la tendencia es que los robots presenten gran flexibilidad tanto en aprendizaje como en interacción con humanos.

Uno de los problemas que caracteriza al aprendizaje por imitación es el problema de correspondencia, que consiste en el mapeo entre el demostrador y el imitador. Si ambos tienen cuerpos similares, entonces una obvia correspondencia es mapear las correspondientes partes del cuerpo: brazo izquierdo del demostrador con brazo izquierdo del imitador, ojo derecho del

demostrador con ojo derecho del imitador. En el caso anterior hay una clara correspondencia para la ejecución de las acciones. Una correspondencia no tiene que ser una relación uno-a-uno, puede ser uno-muchos, muchos-uno o muchos-muchos. Aunque el número de grados de libertad (o por sus siglas DOFs del inglés *Degrees of Freedom*) en las articulaciones de los brazos de un robot sean distintas puede haber correspondencia. Un robot puede imitar a un humano a saludar sin que el número de sus articulaciones sea la misma. El problema de correspondencia trata de encontrar la secuencia de acciones realizadas por el imitador que correspondan a las del demostrador de acuerdo a su propio cuerpo.

En trabajos recientes [Saunders et al., 2007] una propuesta de solución al problema de correspondencia consiste en generar los ejemplos con el mismo robot. A esta técnica le llaman "auto-imitación". Sin embargo, esta técnica tiene las mismas características de "clonación de comportamiento" que es la que utilizamos en esta tesis y que a continuación se describe.

2.4 Clonación de comportamiento

Esta técnica se originó a principios de los noventa a raíz de críticas a la Inteligencia Artificial en el sentido de que no se daba importancia a las habilidades de bajo nivel y sólo se enfocaba a los procesos cognitivos de alto nivel [Michie y Sammut, 1995]. El término clonación de comportamiento, que a lo largo de la tesis referiremos como clonación, fué acuñado por Donald Michie en 1992 y tiene por objetivo extraer conocimiento explícito de habilidades de bajo nivel. Clonación busca hacer explícita una habilidad y que ese conocimiento sirva para construir controladores que reproduzcan esa habilidad. Al ser difícil programar manualmente ese tipo de habilidades, el aprendizaje computacional es de gran ayuda para construir estos sistemas de control.

La técnica básica consiste en obtener ejemplos o trazas de las variables involucradas en el proceso cuando un operador o sistema ejecuta la tarea. Estos ejemplos se dan como entrada a algoritmos de aprendizaje computacional que pueden construir modelos, por ejemplo reglas o árboles, que al ejecutarse producirán comportamientos similares a quien generó los ejemplos. Los modelos resultantes pueden ser incorporados como programas controladores. A estos programas se les llama "clones" porque reproducen la habilidad de una persona o sistema.

Una de las ventajas de esta técnica es que no requiere un modelo matemático de control tradicional pues no siempre se cuenta con la información suficiente o el modelo exacto del proceso. La persona que entrena, únicamente necesita ejecutar la habilidad más no comprender sus fundamentos teóricos. En este contexto, los "clones" son representaciones simbólicas

de comportamientos de bajo nivel.

Los principales objetivos de esta técnica son: (i) producir "clones" que puedan realizar la tarea de control y (ii) producir "clones" que hagan explícita la habilidad, que describan lo que el operador hace [Bratko, 2000]. El primer objetivo es importante porque producir un clon que sirva como controlador facilita la programación al no tenerse que codificar explícitamente la tarea. El segundo objetivo permite tener información explícita sobre la estrategia de control llevada a cabo por el operador. En esta tesis nos interesa cubrir los dos objetivos puesto que buscamos aprender "clones" que sean confiables para el control de un robot móvil y también queremos que los modelos que se obtengan sean fáciles de interpretar y que puedan utilizarse en otras tareas.

La formulación original de la clonación es la recuperación de la estrategia de control a partir de trazas y es un mapeo directo de estados a acciones. Una traza está formada generalmente por pares (Estado, Clase) donde Estado es un vector de atributos $(x_1, x_2, ...)$ y Clase es la acción efectuada por el operador en ese estado. Una de las principales limitantes de esta representación es que carece de estructura y es difícil mostrar la estrategia del operador de forma descriptiva. En el aspecto de control, los "clones" obtenidos no son muy robustos con respecto a cambios en la tarea de control.

Las técnicas de aprendizaje que se han usado más frecuentemente en clonación son los árboles de decisión [Quinlan, 1990], aunque otras técnicas que reconstruyan funciones a partir de ejemplos pueden ser usadas, como las redes neuronales.

Aprendizaje por imitación y clonación son técnicas que tienen objetivos y aplicaciones diferentes. Imitación surgió con el fin de automatizar la programación de robots industriales. La idea principal es que exista un demostrador que realice la tarea que el imitador va a aprender. El demostrador puede ser diferente al imitador. Clonación busca extraer conocimiento explícito de una habilidad para construir controladores que la puedan reproducir. No se originó específicamente para robótica sino que ha sido aplicada en diversos dominios como por ejemplo, aprendizaje de vuelo, control de grúas. La Tabla 2.1 resume las diferencias entre ambas técnicas.

En robótica, el enfoque de adquisición de habilidades a partir de ejemplos se ha utilizado ampliamente. Sin embargo, la abstracción de la información de bajo nivel en conceptos de alto nivel no ha sido ampliamente explotada. Uno de los primeros trabajos que explora este tema en robótica es [Klingspor y Sklorz, 1995].

En esta tesis se utiliza la técnica de clonación para el aprendizaje de habilidades de navegación. Los ejemplos son tomados mientras el usuario guía al robot a realizar una tarea y

	Imitación	Clonación
Inicios	Principios de los 80s.	Principios de los 90s.
Objetivo	Automatizar la programación manual de robots industriales.	(1) Extraer conocimiento explícito de habilidades. (2) Construir controladores que reproduzcan la habilidad.
Obtención de ejemplos	El demostrador puede ser diferente al imitador.	El usuario guía al aprendiz (imitador).
Aplicación	Robótica.	Aprendizaje de vuelo, control de grúas, control de bicicleta, vehículos, robots, etc.

Tabla 2.1: Diferencias entre aprendizaje por imitación y clonación

consisten en lecturas de los sensores, posición del robot y la acción realizada por el usuario. Para el aprendizaje, se lleva a cabo un proceso de abstracción en el cual se transforman las trazas de datos crudos a trazas de alto nivel. La representación en lógica de primer orden permite que las reglas de control (Programas Teleo-Reactivos) que se aprendan sean fácilmente interpretables e integradas a arquitecturas de robots de servicio.

Para el aprendizaje de los programas teleo-reactivos, utilizamos un enfoque de aprendizaje computacional para aprender relaciones a partir de ejemplos: Programación Lógica Inductiva (ILP) que se describe en la siguiente sección.

2.5 Aprendizaje relacional

En esta sección se presentan los fundamentos básicos del aprendizaje relacional. Una revisión más detallada del tema puede verse en [Lavrac y Dzeroski, 1994]. Algunos conceptos de esta sección se basan en [Morales, 2006] y en [Bratko, 1986].

Numerosos algoritmos de aprendizaje inductivo (e.g., inducción de reglas y árboles) usan una representación atributo-valor en la cual los objetos son descritos en términos de atributos fijos. A estos algoritmos se les conoce también como algoritmos proposicionales. Sus dos principales limitaciones son su escasa expresividad en la representación y la reducida capacidad de tomar en cuenta el conocimiento del dominio disponible.

Existe otro tipo de sistemas de aprendizaje que inducen descripciones de relaciones en forma de definiciones de predicados. En estos sistemas los objetos pueden describirse estructuralmente, es decir, en términos de sus componentes y las relaciones entre éllos. La representación de los ejemplos y conocimiento del dominio se hacen en lógica de primer orden. A este tipo de sistemas se les conoce como sistemas de programación lógica inductiva

(ILP).

2.5.1 Formulación del problema de *ILP*

En el aprendizaje relacional la tarea es inducir, a partir de ejemplos, una relación desconocida (el predicado objetivo). El usuario especifica con ejemplos positivos lo que se espera que el programa haga, mientras que agregando ejemplos negativos se indica lo que el programa no debería hacer. Adicionalmente, el usuario puede especificar predicados como conocimiento del dominio que puedan usarse al aprender el programa.

El objetivo de ILP es encontrar una relación o hipótesis ($\mathcal{H}$) que sea completa (*i.e.*, cubre todos los ejemplos positivos) y consistente (*i.e.*, no cubre ningún ejemplo negativo). Una hipótesis es una descripción en la forma de un conjunto de reglas. Si el ejemplo satisface la descripción se dice que el ejemplo es cubierto por la hipótesis. Los predicados correspondientes al conocimiento del dominio determinan el lenguaje en el que se expresará la hipótesis sobre el predicado objetivo. La tarea de ILP puede formularse como sigue:

Dados:

- un conjunto de ejemplos de entrenamiento $\mathcal{E}$ formado por hechos cerrados sin variable positivos $\mathcal{E}^+$ y un conjunto de ejemplos negativos $\mathcal{E}^-$ de un predicado desconocido p,
- conocimiento del dominio $\mathcal{B}$, definiendo predicados que pueden utilizarse en la definición de p y que proporcionan información adicional sobre los argumentos de los ejemplos del predicado p.

Encontrar:

- una hipótesis $\mathcal{H}$ para p, que sea completa y consistente con respecto a los ejemplos $\mathcal{E}$ y al conocimiento del dominio $\mathcal{B}$.

La ventaja de ILP con respecto a los algoritmos de aprendizaje atributo-valor es el lenguaje de hipótesis que proporciona expresividad y la posibilidad de incorporar conocimiento del dominio al proceso de aprendizaje. En los algoritmos atributo-valor por el contrario, el conocimiento del dominio se limita a agregar nuevos atributos. El énfasis en una aplicación de ILP está en hacer una buena representación de los ejemplos y del conocimiento del dominio.

2.5.2 Algoritmo básico

El proceso de inducción puede verse como un proceso de búsqueda en el espacio de hipótesis. Uno de los esquemas más usados en *ILP* es el enfoque de búsqueda de lo general a lo específico (*top-down*), que inicia con las hipótesis muy generales que se especializan repetidamente hasta que no cubran ejemplos negativos (o cubra muy pocos). Cada refinamiento toma una hipótesis H_1 y genera una hipótesis más específica H_2 de tal manera que H_2 cubra un subconjunto de casos cubiertos por H_1. El espacio de hipótesis y sus especializaciones forman un grafo de refinamiento.

El algoritmo básico tiene dos ciclos: el externo de cobertura y el interno de especialización. El ciclo de cobertura se encarga de la construcción de la hipótesis y el de especialización realiza la construcción de las cláusulas. El Algoritmo 2.1 muestra los pasos básicos de un algoritmo de lo general a lo específico (*top-down*) que supone una búsqueda *hill-climbing* en el grafo de refinamiento. A continuación se describe el Algoritmo 2.1 mostrando entre paréntesis los pasos a los que corresponde la explicación.

Algoritmo 2.1 Algoritmo genérico de *ILP* (general a específico)

Entrada: Conjunto de ejemplos positivos y negativos $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$, conocimiento del dominio $\mathcal{B}$
Variables locales: cláusula c a refinar
Salida: Hipótesis $\mathcal{H}$

1: Inicializar $\mathcal{E}_a = \mathcal{E}$
2: Inicializar $\mathcal{H} = \emptyset$
3: **repeat** {cobertura}
4: Inicializar $c = p(X_1, ..., X_n) \leftarrow$ {La cláusula más general}
5: **repeat** {especialización}
6: Encuentra el mejor refinamiento $c_m \in \rho(c)$
7: $c = c_m$
8: **until** criterio de necesidad se satisface (paro) {deja de agregar literales a c ($\mathcal{B} \cup \mathcal{H} \cup c$ es consistente)}
9: Agrega c a $\mathcal{H}$ para obtener una nueva hipótesis $\mathcal{H}' = \mathcal{H} \cup c$
10: Eliminar de $\mathcal{E}_a$ ejemplos positivos cubiertos por c para obtener un nuevo conjunto de entrenamiento $\mathcal{E'}_a = \mathcal{E}_a - cubre(\mathcal{B}, \mathcal{H}', \mathcal{E}_a^+)$
11: $\mathcal{E}_a = \mathcal{E'}_a$
12: $\mathcal{H} = \mathcal{H}'$
13: **until** criterio de suficiencia se satisface {deja de agregar cláusulas a $\mathcal{H}$ ($\mathcal{B} \cup \mathcal{H} \cup c$ es completa)}
14: **Return:** Hipótesis $\mathcal{H}$

- (1,2,3) El ciclo externo (cobertura) inicia con el conjunto completo de ejemplos de entrenamiento y un conjunto vacío de cláusulas.

Algoritmo 2.2 Función $cubre(\mathcal{B}, \mathcal{H}', \mathcal{E}_a^+)$

Entrada: Conjunto de ejemplos positivos $\mathcal{E}^+$, conocimiento del dominio $\mathcal{B}$, Hipótesis $\mathcal{H}'$
Salida: Nuevo conjunto de ejemplos ${\mathcal{E}'}_a$
1: ${\mathcal{E}'}_a = \mathcal{E}_a - cubre(\mathcal{B}, \mathcal{H}', \mathcal{E}_a^+)$ {Eliminar de $\mathcal{E}_a$ ejemplos positivos cubiertos por c }
2: **Return:** Nuevo conjunto de entrenamiento ${\mathcal{E}'}_a$

- (4) El ciclo interno (especialización) inicia con una cláusula muy general (sin condiciones en el cuerpo).
- (5,6,7,8) En el ciclo interno (especialización) se construyen cláusulas mediante una búsqueda heurística en el espacio de posibles cláusulas. A la cláusula más general se le van agregando literales (condiciones) hasta que sólo cubre ejemplos positivos y ningún negativo (la cláusula es consistente). Ésto se hace mediante el uso de operadores de especialización, también llamados operadores de refinamiento. Las operaciones básicas que realizan en las cláusulas son: (i) sustituir variables por términos y (ii) agregar literales al cuerpo de la cláusula. Cuando se encuentra una cláusula que cubre con el criterio de necesidad (*i.e.*, no cubrir negativos), se deja de especializar.
- (9) Se agrega a la hipótesis la cláusula que cumplió con el criterio de necesidad en el ciclo de especialización.
- (10) Se eliminan los ejemplos positivos que la cláusula agregada a la hipótesis cubre.
- (11) El nuevo conjunto de entrenamiento se reduce al eliminar ejemplos.
- (12) Se actualiza la hipótesis pues ahora tiene una cláusula nueva.
- (13) El ciclo se repite hasta que todos los ejemplos positivos son cubiertos (la hipótesis es completa), donde:

  Dado cierto conocimiento del dominio $\mathcal{B}$, una hipótesis $\mathcal{H}$ y un conjunto de ejemplos positivos $\mathcal{E}_a^+$, la hipótesis $\mathcal{H}$ cubre el ejemplo $e \in \mathcal{E}$ con respecto al conocimiento del dominio $\mathcal{B}$ si $\mathcal{B} \cup \mathcal{H} \vDash e$. Consecuentemente, la función $cubre(\mathcal{B}, \mathcal{H}', \mathcal{E}_a^+)$ se define como:

  $$cubre(\mathcal{B}, \mathcal{H}', \mathcal{E}_a^+) = \{e \in \mathcal{E}_a^+ | \mathcal{B} \cup \mathcal{H} \vDash e\}$$

  la función $cubre(\mathcal{B}, \mathcal{H}', \mathcal{E}_a^+)$ establece que e es cubierto por $\mathcal{H}$ dado un conocimiento del dominio $\mathcal{B}$ si e es consecuencia lógica de $\mathcal{B} \cup \mathcal{H}$.

- (14) La salida es una hipótesis completa y consistente.

En dominios con datos perfectos, el criterio de necesidad requiere consistencia, es decir, no cubrir ningún ejemplo negativo. El criterio de suficiencia requiere completez que significa cubrir todos los ejemplos positivos. Cuando existen datos con ruido, los criterios de completez y consistencia se relajan y los criterios se basan en el número de ejemplos positivos y negativos cubiertos por una cláusula o hipótesis.

2.5.3 Ejemplo de aprendizaje

En esta sección se muestra un ejemplo en el que se aplica el método de inducción descrito en la sección anterior. El ejemplo está tomado de [Bratko, 1986] aunque la forma de solución es distinta a la mostrada en la referencia.

Notación. A lo largo de la tesis se utiliza una notación similar a la sintaxis de Prolog: una variable es una cadena que inicia con una letra mayúscula, las constantes son cadenas de minúsculas, los nombres de predicados son cadenas de minúsculas seguidas de un paréntesis con argumentos de entrada/salida, el símbolo: "_" representa a una variable anónima y se usa cuando se desea ignorar algún valor que no es de interés. Ejemplo:

$$\begin{aligned} &p(Arg_1, Arg_2) \leftarrow \\ &\quad cond1(Arg_1, Arg_3), \\ &\quad cond2(Arg_1, \_, Arg_4)\,. \end{aligned}$$

donde p es el nombre de un predicado con dos argumentos: Arg_1 y Arg_2. El cuerpo del predicado está formado por las literales $cond_1$ y $cond_2$ cuya premisa es el cuerpo de la cláusula y cuyo consecuente es la cabeza.

En la Figura 2.3 cada nodo es *progenitor* del nodo inferior al que apunta. Así, `pam` y `tom` son *progenitor*es de `bob` y `liz`; `bob` es padre de `ann` y `pat` y así sucesivamente. Se quiere encontrar una definición para el predicado $tiene\_hija(X)$ en términos de los predicados $progenitor(X, Y)$, $hombre(X)$ y $mujer(Y)$.

El predicado objetivo especifica las literales que el sistema de *ILP* puede usar para construir las cláusulas. El predicado objetivo está formado por una cabeza cuyos argumentos pueden ser: de entrada (+argumento), de salida (-argumento) o constantes (#argumento). El cuerpo está formado por los predicados que pueden aparecer en la hipótesis. No necesariamente el predicado aprendido incluirá todos los predicados dados por el usuario. Por

ejemplo, en el predicado objetivo *tiene_hija*, las literales $hombre(+b)$ y $mujer(+b)$ indican que el predicado puede usarlas para construir la cláusula, no que deban estar ambos incluidos. La notación que se muestra es la usada en ALEPH [Srinivasan-ALEPH-url, 1999]. A continuación se muestra el predicado objetivo:

$$
\begin{aligned}
&tiene\_hija(+a). \;\; \%\, Encabezado \;\; del \;\; predicado \;\; objetivo \\
&\quad progenitor(+a,-b), \;\; \%\, Posible \;\; literal \\
&\quad hombre(+b)\,, \;\; \%\, Posible \;\; literal \\
&\quad mujer(+b)\,. \;\; \%\, Posible \;\; literal
\end{aligned}
$$

Los ejemplos positivos son instancias del predicado objetivo a partir de las relaciones familiares mostradas en el grafo: (*e.g.*, *tiene_hija*(`tom`) es verdadero pues `tom` tiene una hija llamada `liz`).

Los ejemplos negativos son instancias falsas como por ejemplo *tiene_hija*(`pam`) no es verdadero porque `pam` no tiene una hija sino un hijo llamado `bob`. Los ejemplos positivos y negativos se muestran en la Tabla 2.2.

El conocimiento del dominio está definido por los predicados $progenitor(X,Y)$, $mujer(Y)$ y $hombre(Y)$ (ver Tabla 2.3). Básicamente los predicados describen las relaciones familiares mostradas en el grafo.

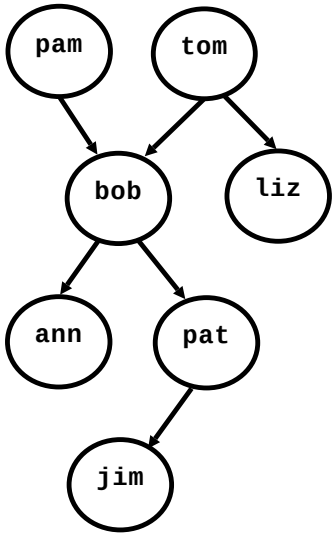


Figura 2.3: Relaciones familiares

Ejemplos positivos	Ejemplos negativos
tiene_hija(tom).	*tiene_hija*(pam).
tiene_hija(bob).	*tiene_hija*(jim).
tiene_hija(pat).	

Tabla 2.2: Ejemplos positivos y negativos para aprender el predicado $tiene\_hija(X)$

progenitor(pam,bob).	*mujer*(pam).
progenitor(tom,bob).	*mujer*(liz).
progenitor(tom,liz).	*mujer*(ann).
progenitor(bob,ann).	*mujer*(pat).
progenitor(bob,pat).	*mujer*(eve).
progenitor(pat,jim).	*hombre*(tom).
progenitor(pat,eve).	*hombre*(bob).
	hombre(jim).

Tabla 2.3: Conocimiento del dominio para aprender el predicado *tiene_hija*(X)

Una vez definido el problema de aprendizaje, se construirá una hipótesis usando el Algoritmo 2.1. Como criterio de necesidad se definió que una cláusula se acepta si no cubre ejemplos negativos, mientras que como criterio de suficiencia se determinó que una cláusula se agrega a la hipótesis si cubre al menos 2 ejemplos positivos. La Figura 2.4 ilustra parte del proceso. Se inicia por el predicado más general que cubre tanto los ejemplos positivos como los negativos (*i.e.* 3 ejemplos positivos y 2 ejemplos negativos). La primer especialización agrega la literal $progenitor(X, Y)$ que cubre los 3 ejemplos positivos pero 1 negativo por lo que no es aceptada pues no cumple con el criterio de necesidad. En la siguiente especialización se agrega la literal $hombre(Y)$ pero cubre solamente 2 ejemplos positivos y también cubre 1 negativo. En la última especialización se agrega la literal $mujer(Y)$ que cumple con el criterio de necesidad pues no cubre ningún negativo y también cumple con el criterio de suficiencia por lo que se agrega a la hipótesis. Para este caso, la hipótesis está formada por una sola cláusula que es completa y consistente para describir al conjunto de ejemplos.

La cláusula aceptada es:

$$
\begin{aligned}
&tiene\_hija(X) \ :- \\
&\quad progenitor(X, \ Y), \\
&\quad mujer(Y).
\end{aligned}
$$

Ejemplos de sistemas de *ILP*. La Tabla 2.4 muestra algunos de los sistemas de *ILP* más conocidos: ALEPH [Srinivasan-ALEPH-url, 1999] emula algunas funcionalidades de varios sis-

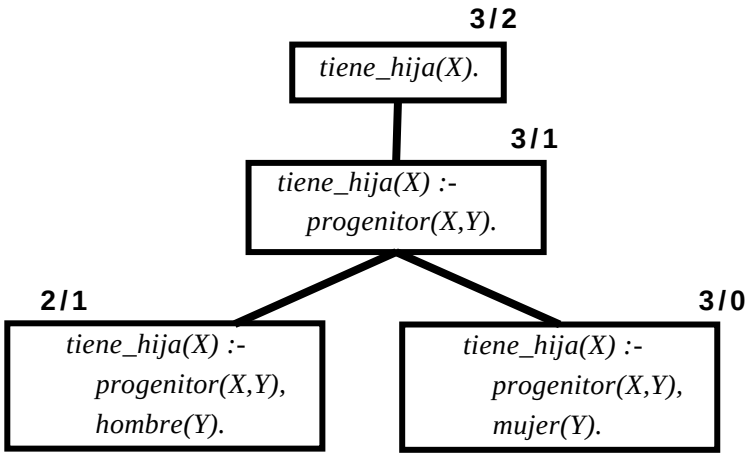


Figura 2.4: La mejor cláusula es aceptada

temas, permitiendo seleccionar la estrategia de búsqueda. LINUS [Lavrac y Dzeroski, 1994] transforma el problema de *ILP* a una representación proposicional y la inducción es ejecutada por un algoritmo atributo-valor, es decir, si usa CN2 usará *beam search*. Sistemas como Progol [Muggleton, 1996] o FOIL [Quinlan y Cameron-Jones, 1993] pueden aprender rangos con datos numéricos usando predicados como $is, >, <, =$, mientras que FORS [Karalič y Bratko, 1997] aprende funciones de regresión lineal. TILDE [Blockeel y Raedt, 1998] permite el aprendizaje de árboles de decisión relacionales. HYPER [Bratko, 1999], a diferencia de los anteriores, especializa hipótesis completas en vez de efectuar una búsqueda de cláusulas individuales.

Sistema	Búsqueda
Progol	A*
FOIL	*Hill Climbing*
TILDE	*Hill Climbing*
ALEPH	Varios
FORS	*Beam search*
LINUS	Depende del algoritmo
HYPER	*Best First*

Tabla 2.4: Sistemas de *ILP* y sus estrategias de búsqueda

En esta tesis estamos usando ALEPH (*A Learning Engine for Proposing Hypotheses*) para la primer fase del aprendizaje. ALEPH ha sido ampliamente usado en diversas aplicaciones de *ILP* y proporciona una plataforma flexible de pruebas. ALEPH selecciona un ejemplo del

conjunto dado y construye la cláusula más específica seleccionando un conjunto de predicados del conocimiento del dominio que sean verdaderos para ese ejemplo. ALEPH busca un subconjunto de literales de la cláusula más específica que sea más general. La cláusula que cubre el máximo número de ejemplos positivos y el mínimo número de ejemplos negativos se agrega a la hipótesis. Los ejemplos cubiertos por la cláusula se eliminan. El proceso se repite hasta que se han cubierto todos los ejemplos positivos.

En esta tesis se usa aprendizaje relacional: (i) en el aprendizaje de tareas para un robot móvil y (ii) reconocimiento de ademanes a partir de secuencias expresadas en lógica de primer orden. Las principales ventajas y desventajas del enfoque relacional son las siguientes:

Ventajas:

- Es útil para el aprendizaje de descripciones de estados y modelos de acciones pues facilita la representación de relaciones entre objetos.
- Permite la integración de conocimiento del dominio en forma de hechos o predicados a diferencia del enfoque proposicional que tiene una representación atributo-valor limitada.
- El nivel de expresividad es alto por lo que las tareas de alto nivel se expresan de forma más natural que como podrían representarse usando algoritmos proposicionales.

Desventajas:

- Dificultades en el manejo de grandes cantidades de datos.
- La definición del conocimiento del dominio y predicados a aprender no siempre es fácil.
- Carece de facilidades numéricas. La aritmética tiene que darse de forma explícita (e.g. *menor_que, mayor_que*).
- Los algoritmos tienden a ser más lentos y requerir más memoria que los algoritmos proposicionales.

Mediante el uso de clonación, una representación relacional y programación lógica inductiva, en esta tesis se aprenden habilidades en forma de programas lógicos. Los programas aprendidos tienen ciertas características que se fundamentan en un formalismo conocido como teleo-reactivo, que es el tema a revisar en la siguiente sección.

2.6 Programas Teleo-Reactivos (PTRs)

Nils Nilsson (1994) introduce el formalismo teleo-reactivo con la idea de proporcionar a robots móviles un control reactivo y adaptable a ambientes dinámicos. La idea general de los PTRs es simple y busca representar de forma compacta comportamientos difíciles. Una secuencia teleo-reactiva o PTR es una lista ordenada de reglas de producción como la siguiente:

$$\begin{aligned} K_1 &\rightarrow a_1 \\ K_2 &\rightarrow a_2 \\ &\cdots \\ K_m &\rightarrow a_m \end{aligned}$$

Las K_i son condiciones que son evaluadas con respecto a las lecturas de los sensores y al modelo del mundo y las a_i son las acciones definidas en ese mundo. Las acciones pueden ser: (i) básicas como por ejemplo, `girar-izquierda` para un robot móvil o `patear` para un jugador de futbol y (ii) llamadas a otro PTR; en este caso es un PTR jerárquico.

Las reglas son recorridas de arriba hacia abajo y son evaluadas contínuamente. Para la primer condición que se satisface se ejecuta la acción correspondiente. Sin embargo, la diferencia con los sistemas convencionales de reglas de producción es que las acciones pueden ser continuas, por ejemplo, girar hacia la izquierda a cierta velocidad indefinidamente. En el caso de un robot móvil existe una gran diferencia entre la acción moverse hacia adelante de forma continua y la acción discreta moverse hacia adelante un metro. En un PTR una acción continua se ejecuta mientras la condición que se está cumpliendo siga siendo verdadera. Las condiciones deben ser evaluadas contínuamente. Acciones discretas como detenerse también pueden usarse.

Generalmente, los PTRs se organizan de tal manera que K_1 es la condición meta, a_1 es la acción nula y K_m es verdadera, es decir, la acción que se ejecuta si ninguna de las condiciones previas de la lista se cumple. Las reglas al inicio de la lista son las relevantes cuando el robot está cerca de lograr una meta particular. Las reglas al final de la lista son las que se ejecutan cuando el robot está lejos de su meta. La idea es que las reglas de la parte inferior de la lista disparen acciones que satisfagan las condiciones de reglas de la parte superior.

> "Una secuencia teleo-reactiva es un programa de control de agentes que conduce al agente hacia la meta (de ahí *teleo*) de forma que tome en cuenta los cambios del ambiente (de ahí *reactivo*)".

En este contexto, como agente se entiende a sistemas que perciben y modelan su ambiente y ejecutan acciones para lograr metas.

Propiedades. Un PTR satisface la propiedad de **regresión** si las condiciones son cada vez más específicas conforme la prioridad es más alta. Ésto significa que cada condición dispara una acción que hace que se cumpla una condición de mayor prioridad. Un PTR es completo si las condiciones cubren todos los casos. Si un PTR satisface la propiedad de **regresión** y es completo, entonces es **universal**. Un PTR **universal** siempre alcanzará su condición meta a partir de cualquier estado inicial si no existieran errores de sensado o de ejecución.

Representación gráfica. Un PTR puede representarse también como un grafo. La Figura 2.5 [Nilsson, 1994] muestra la representación gráfica de un PTR simple. Para ejecutar la versión gráfica del programa se busca el nodo de menor profundidad que sea verdadero y ejecutando la acción etiquetada en el arco que sale del nodo. Se considera que la condición meta es la raíz (K_1). Si existe más de una acción que conduzca a una condición entonces se tiene un árbol en vez de un grafo con un solo camino.

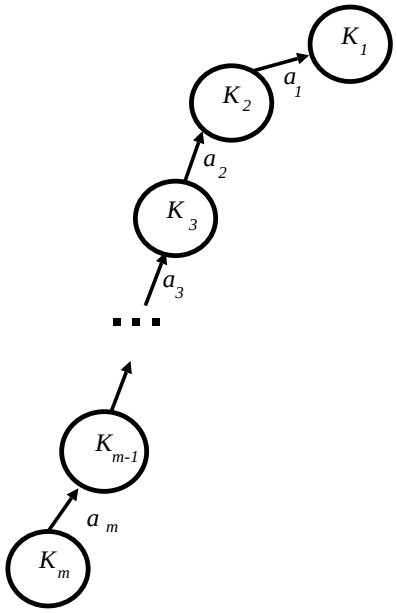


Figura 2.5: Representación gráfica de un PTR [Nilsson, 1994]. Los nodos representan las condiciones donde K_1 representa la condición meta y K_m la condición que siempre es verdadera. Los arcos son las acciones. Se busca el nodo de menor profundidad que sea verdadero y se ejecuta la acción que sale del nodo.

En los orígenes de este formalismo, los primeros experimentos se hicieron programando PTRs de forma manual para robots simulados y para un robot Nomadic con 16 sensores in-

frarojos y 16 sonares. En el robot Nomadic se probaron las tareas de seguimiento de pared y desplazamiento por un pasillo con buenos resultados. Posteriormente se realizó la primer implementación de estas ideas integrando planeación y aprendizaje en una plataforma de simulación; este trabajo [Benson y Nilsson, 1995] se describe en el Capítulo 3. Aunque los resultados iniciales mostraron resultados promisorios, existen pocos trabajos que exploren estas ideas y las extiendan. Recientemente, desarrolladores de juegos han integrado comportamiento teleo-reactivo a sus personajes [Kochenderfer, , Champandard, 2007]. En esta tesis exploramos estos conceptos. Sin embargo, a diferencia del formalismo básico, las reglas que se aprenden en este trabajo, tienen más de una condición para ejecutar una acción.

A continuación se muestra la terminología que se usará a lo largo de la tesis:

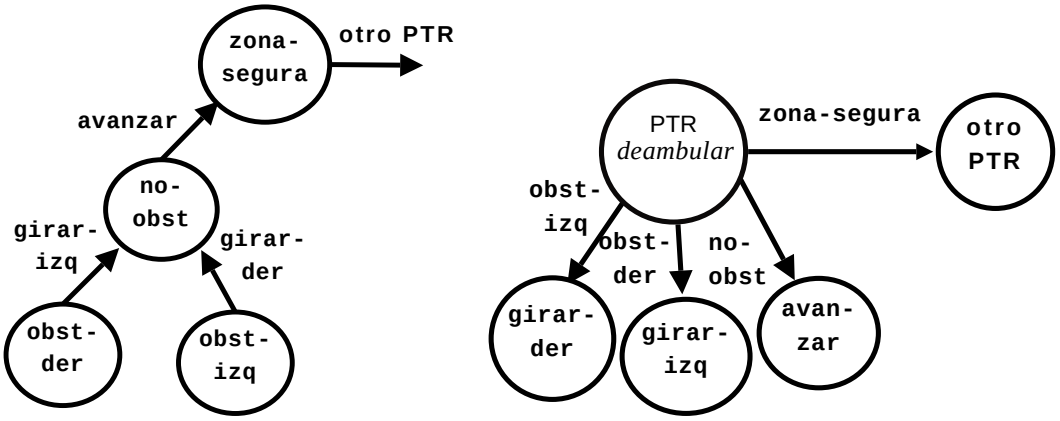
Acciones de bajo-nivel o atómicas. Llamamos así a las instrucciones de movimiento como por ejemplo, `girar_izquierda, avanzar`. A lo largo de la tesis se les llama simplemente acciones.

PTRs básicos. Definimos así a aquéllos PTRs que necesitan un conjunto de dos o más acciones de bajo-nivel o atómicas como por ejemplo, ***deambular***.

PTRs jerárquicos. Son aquéllos que además de acciones de bajo-nivel incluyen en su conjunto de acciones a otros PTRs, por ejemplo, *ir-a*.

Representación gráfica. Un PTR puede representarse también como un árbol TR [Nilsson, 1994], cuyos nodos corresponden a las condiciones y los arcos a las acciones. En la figura 2.6(a) se muestra el árbol TR para el programa ***deambular***. Para ejecutarlo, se busca el nodo verdadero menos profundo y se ejecuta la acción que indica el arco que sale de ese nodo. Cada acción normalmente logra la condición a la que entra ese arco. En este caso se muestra un PTR básico para ***deambular***. Existen dos arcos que pueden conducir a hacer verdadero el nodo sin-obstáculo: cuando se ejecutan las acciones `girar-izquierda` o `girar-derecha`. Si no existen obstáculos entonces el robot puede `avanzar`, logrando estar en una zona-segura. Una vez estando en una *zona segura* puede ejecutarse otro PTR.

Para nuestro trabajo, una representación gráfica más adecuada que la de [Nilsson, 1994] la vemos en la figura 2.6(b). Los nodos representan acciones de bajo-nivel, PTRs básicos o PTRs jerárquicos, y los arcos se etiquetan con las condiciones que se necesitan cumplir para que se ejecute cada acción. Este grafo muestra cómo se va construyendo la jerarquía de acciones. El nodo `avanzar` puede utilizarse como liga a otros PTRs, como se verá en las siguientes secciones.



(a) Árbol teleo-reactivo para *deambular* (b) Árbol de acciones para *deambular*

Figura 2.6: (a) Árbol teleo-reactivo: los nodos representan la condición y los arcos indican la acción, (b) Árbol de jerarquías de acciones: los nodos representan las acciones de bajo-nivel, PTRs básicos y PTRs jerárquicos. Los nodos representan las condiciones

En esta tesis se exploran y se extienden estos conceptos en un robot real por lo que se tienen dificultades no consideradas en el formalismo básico. Una de las dificultades que no se contemplan en el sistema implementado basado en el formalismo teleo-reactivo [Benson, 1996] es que no se usan robots reales por lo que el sistema no aborda el problema de la representación de los cientos de datos recibidos por los sensores. Por otra parte, la tecnología de robots móviles a mediados de la década de los 90s no estaba suficientemente avanzada para probar el sistema. Otro aspecto que en esta tesis se extiende es el aprendizaje de PTRs jerárquicos. En esta tesis:

- Se incorpora un procesamiento para transformar información de bajo nivel de los sensores a una representación del mundo de alto nivel.
- Se aprenden PTRs básicos utilizando clonación.
- Se aprenden PTRs jerárquicos utilizando inducción de gramáticas.
- Se prueban los PTRs aprendidos en un robot real, integrándose a una arquitectura para robots de servicio.

Para el aprendizaje de PTRs compuestos o jerárquicos se desarrolló un algoritmo que induce gramáticas a partir de secuencias de acciones o PTRs básicos. Estas gramáticas son en sí programas lógicos. Se les conoce como gramáticas de cláusulas definidas (GCDs) y se describen en la siguiente sección.

2.7 Gramáticas de Cláusulas Definidas (GCDs)

Una secuencia de acciones tiene implícita una estructura que puede proporcionar información para la construcción de programas compuestos. Esta estructura puede aprenderse mediante la inducción de la gramática que describe las secuencias.

Una gramática es un conjunto finito de reglas llamadas producciones que especifican un lenguaje y puede usarse para: (i) generar cadenas válidas del mismo, (ii) reconocer si una cadena dada pertenece al lenguaje, y (iii) describir la estructura de las cadenas. Las gramáticas libres de contexto (CFGs) han sido usadas ampliamente, especialmente para procesamiento de lenguaje natural y son la base para la sintaxis de la mayoría de lenguajes de programación. Las CFG son gramáticas en las que cada regla de producción tiene la forma

$$S \rightarrow a$$

donde S es un símbolo no-terminal y a es una cadena de terminales y/o no-terminales. Una CFG es una 5-tupla (P, A, N, T, S) donde

- P es un conjunto de reglas de producción de la forma $X \rightarrow \beta$ donde X es un miembro de N y β es una cadena sobre el alfabeto A,
- A es un alfabeto de símbolos,
- N es un conjunto de símbolos no-terminales,
- T es un conjunto de símbolos terminales ($N \cup T = A$),
- S es un símbolo no-terminal distinguido conocido como el símbolo de inicio.

En esta tesis nos enfocamos al aprendizaje de gramáticas de cláusulas definidas (GCDs) [Robertson, 1998, Blackburn y Striegnitz, 2002], y [Bratko, 1986] que son una extensión de las gramáticas libres de contexto, las cuales pueden tener argumentos y son expresadas y ejecutadas en Prolog. Las GCDs se usan generalmente para análisis sintáctico o generación de una lista de símbolos de acuerdo a la gramática. Su principal aplicación es el procesamiento de lenguaje natural.

La forma general de una cláusula de GCDs es la siguiente:

`Cabeza` → `Cuerpo`

que significa: "una posible forma de Cabeza es Cuerpo", donde Cabeza y Cuerpo son términos de Prolog. Un término puede ser una constante, una variable o un término compuesto, por ejemplo: $frase(sustantivo(\texttt{juan}), resto(verbo(\texttt{come}), sustantivo(\texttt{manzanas})))$.

- Cabeza se forma por un símbolo no-terminal.
- Cuerpo puede contener cualquier secuencia de símbolos terminales o no-terminales, separados por una coma (,) indicando una conjunción y/o un punto y coma (;) indicando una disyunción.
- En el cuerpo de una cláusula de GCDs, los símbolos terminales se distinguen de los no-terminales por estar representados como una lista de Prolog: $[T_1, T_2, \ldots T_n]$.
- Pueden agregarse condiciones adicionales en la forma de llamadas a procedimiento de Prolog en el cuerpo de la cláusula de GCDs. Se encierran en llaves: $\{P_1, P_2, \ldots P_n\}$.

Las GCDs son una abreviación de cláusulas ordinarias de Prolog. Cada cláusula GCD se traduce a una cláusula estándar de Prolog al ser cargada. De esta forma, Prolog convierte las reglas de la gramática dadas en un programa para reconocer y generar secuencias. Los argumentos requeridos para las listas de entrada y salida no se escriben explícitamente en una regla gramatical pero se agregan cuando la regla se traduce a una cláusula ordinaria. Una gramática escrita en GCDs es ya un programa de análisis sintáctico para esa gramática.

Supongamos que un robot cuenta con dos posibles acciones: `avanzar`, que consiste en moverse 0.5 m hacia adelante y `retroceder` para moverse 0.5 m hacia atrás. Cada uno de esos desplazamientos es un paso. Un movimiento es una secuencia de pasos y puede consistir en un paso o un paso seguido de un movimiento. Ésto puede expresarse con la gramática libre de contexto (CFG) siguiente:

```
Movimiento → Paso
Movimiento → Paso,Movimiento
Paso → avanzar
Paso → retroceder
```

donde `Movimiento` y `Paso` son símbolos no-terminales por lo que inician con mayúscula. Los símbolos terminales `avanzar` y `retroceder` se escriben con minúsculas.

Expresando la gramática libre de contexto anterior como una GCD se obtiene el siguiente código de Prolog:

```
movimiento --> paso.
movimiento --> paso,movimiento.
paso --> [avanzar].
paso --> [retroceder].
```

donde los símbolos no terminales se escriben en minúsculas y los símbolos terminales se escriben entre corchetes. Esta gramática puede ser usada para reconocer secuencias como: `avanzar,avanzar,retroceder`.

Una ventaja de las GCDs es que se pueden agregar argumentos a las reglas. Supongamos que las acciones avanzar y retroceder pueden realizarse a diferentes velocidades como por ejemplo: rápido, medio, lento. Es posible incorporar a las reglas argumentos que describan estos atributos. La siguiente GCD muestra que los posibles pasos que la gramática aceptará serán: `avanzar(veloz)` y `retroceder(lento)`.

```
movimiento --> paso.
movimiento --> paso,movimiento.
paso --> [avanzar(veloz)].
paso --> [retroceder(lento)].
```

En esta tesis, el aprendizaje de GCDs tiene dos funciones:

1. A partir de secuencias, se aprenden GCDs como programas lógicos que se usan como controladores.

2. Clasificadores de secuencias. La salida son programas lógicos que analizan sintácticamente secuencias.

En el Capítulo 5 se presenta un algoritmo para aprender GCDs a partir de secuencias de predicados.

2.8 Resumen

En este capítulo se presentaron los conceptos y técnicas que se usan a largo de la tesis. Se describieron los fundamentos de clonación, aprendizaje relacional, el formalismo teleoreactivo y las gramáticas de cláusulas definidas (GCD). Con estos fundamentos se desarrolla

una metodología de aprendizaje basada en clonación y aprendizaje relacional. El objetivo es aprender PTRs para un robot móvil que lo provean de habilidades para realizar tareas en un ambiente dinámico.

Al robot se le instruye por clonación a realizar las tareas. Se está utilizando una representación basada en lógica de primer orden, por lo que usamos *ILP* ya que permite la incorporación de conocimiento del dominio y da una mayor expresividad.

Para aprender PTRs compuestos aprendemos gramáticas (GCDs) que permiten incorporar PTRs aprendidos y crear una jerarquía de conceptos.

En el siguiente capítulo se presenta una revisión del estado del arte de trabajos estrechamente relacionados con estos tópicos.

3

Estado del Arte

3.1 Introducción

En este capítulo se describen los trabajos relacionados con esta investigación. Las técnicas de aprendizaje que se abordan se utilizan en diversas áreas por lo que se seleccionaron los trabajos relevantes que influyen directamente en el desarrollo del trabajo que se presenta en esta tesis. En algunos casos hay trabajos que combinan algunas de éllas. En los utlimos años se han desarrollado numerosos trabajos en aprendizaje. Una revisión exhaustiva de éllos está fuera de los objetivos de la tesis.

3.2 Clonación

La clonación es una técnica para extraer modelos de habilidades de control humanas y su validez ha sido probada en distintos dominios. Este término fue utilizado por primera vez por Donald Michie y consiste básicamente en obtener trazas del comportamiento de un experto humano al desempeñar una tarea de control. Posteriormente esta traza es procesada por un algoritmo de aprendizaje obteniendo un modelo que es insertado en un sistema controlador. Bajo este esquema, clonación se ha utilizado para el control de una grúa [Urbancic y Bratko, 1994], de un avión [Bain y Sammut, 1999] y del péndulo invertido [Michie y Sammut, 1995]. Lograr construir modelos de las habilidades de una persona produce grandes beneficios ya que dichos modelos permiten la construcción de controladores automáticos [Camacho, 1995] o analizar el comportamiento de los operadores mediante una descripción simbólica.

En robótica, una aplicación la encontramos en [D'Este et al., 2003] en la que un robot

móvil Pioneer II-DX aprendió a evadir obstáculos y a perseguir un objetivo. El proceso de clonación consistió en obtener trazas de las acciones hechas por un operador humano al guiar al robot con un *joystick*. Se experimentó con dos algoritmos: C4.5 y una red neuronal de retropropagación, para crear una base de conocimiento que se usó para controlar al robot automáticamente. Los comandos del *joystick* se discretizaron para obtener los valores de clase, por ejemplo: `easy-right`, `hard-left`. Para la tarea de persecución de un objetivo se necesitó agregar información para almacenar la última posición del objetivo en caso de perderlo de vista. Los modelos obtenidos con árboles de decisión mostraron un mejor comportamiento que con la red neuronal.

Otra aplicación [Kadous et al., 2006] muestra cómo enseñar a un vehículo a atravesar un terreno agreste utilizando árboles de decisión. En ambos casos, una sola tarea es ejecutada utilizando un esquema proposicional, originando clones frágiles. En robótica, sin embargo, es común tener múltiples metas y es deseable que las habilidades aprendidas puedan usarse en dominios distintos, lo que no es posible en los trabajos descritos previamente.

La formulación básica de clonación es muy limitada y difícilmente se pueden obtener clones robustos y confiables que permitan un adecuado control en la realización de tareas que involucren manejo de metas. El enfoque básico ha ido evolucionando y se han incorporado mejoras tales como la descomposición del aprendizaje y la idea de un estilo de control teleo-reactivo [Bain y Sammut, 1999]. En trabajos posteriores se introdujo el manejo de metas [Suc y Bratko, 1997, Isaac y Sammut, 2003] y se mostró experimentalmente que existen mejoras significativas, especialmente en la robustez de los clones. Sin embargo, los resultados muestran que la representación carece aún de la estructura necesaria para que el manejo de metas sea más cercano a un control teleo-reactivo. En ninguna de las aplicaciones se aprende la descomposición o jerarquías de acciones que pueden existir en la realización de una tarea. Adicionalmente, clonación no ha sido ampliamente probada en robots móviles.

En resumen, la clonación presenta las siguientes ventajas:

- Se ha aplicado con éxito para control en dominios complejos que manejan valores continuos.
- Los experimentos realizados con robots reales han mostrado buenos resultados aunque han sido limitados debido a la representación proposicional utilizada.
- Permite obtener reglas de comportamiento que difícilmente pueden programarse de forma manual.

- Es un método relativamente fácil de aplicar.

Por otro lado, presenta las siguientes desventajas:

- Los clones obtenidos pueden ser frágiles a pequeños cambios en los parámetros del ambiente.
- Es necesario obtener un gran número y diversidad de ejemplos para obtener buenos resultados.

Contribución. En esta tesis se implementa un marco de clonación en el que se aprenden Programas Teleo-Reactivos (PTRs) para robots móviles. Se utiliza una representación en lógica de primer orden y un sistema de programación lógica inductiva para aprender los predicados de control para el robot. Esta representación facilita ordenar los predicados de acuerdo al enfoque teleo-reactivo. El problema de aprendizaje se dividió en dos partes: (i) aprendizaje de PTRs básicos y (ii) aprendizaje de PTRs compuestos. Los PTRs que se aprenden son fáciles de interpretar, y pueden utilizarse en diferentes tareas y ambientes tener repetir el aprendizaje.

3.3 Aprendizaje de Programas Teleo-Reactivos (PTRs)

En esta tesis los modelos que se aprenden se apegan al formalismo teleo-reactivo propuesto por Nilsson [Benson y Nilsson, 1995]. Como se describió en la sección 2.6, un programa teleo-reactivo (PTR) dirige al robot hacia una meta tomando en cuenta los eventos que ocurren en un ambiente dinámico.

En los trabajos sobre PTRs encontramos dos tipos de enfoque: (i) PTRs aprendidos automáticamente y (ii) PTRs codificados manualmente. TRAIL [Benson y Nilsson, 1995] es la primera aplicación usando el marco teleo-reactivo, integrando aprendizaje, planeación y múltiples metas. Este sistema aprende modelos de acciones o teleo-operadores que pueden ser usados por un planeador automático para construir árboles teleo-reactivos. Los dominios de prueba fueron: (i) *Botworld*, un simulador de un ambiente de construcción en el que existen barras, ensambles de barras y obstáculos y el objetivo era que cada robot recogiera barras, las moviera y conectara a otras barras y estructuras, (ii) simulación de robot asistente en una oficina cuyas tareas eran: entregar mensajes y objetos, sacar copias y entregarlas, y (iii) simulador de avión Cessna cuyas tareas incluían despegar, mantener el nivel de vuelo, navegación y aterrizaje. TRAIL incorpora:

- **Comportamiento teleo-reactivo.** Permite al robot reaccionar apropiada y rápidamente a eventos del ambiente, como por ejemplo evitar obstáculos o recargar su batería.

- **Atención a múltiples metas.** Se refiere a la capacidad del robot para efectuar acciones en el momento en que se presenten las condiciones para hacerlo, no necesariamente siguiendo un orden pre-establecido. Si el robot tiene que realizar 2 tareas y al estar ejecutando acciones de la tarea 1 se cumplen condiciones de la tarea 2 el robot ejecuta las acciones correspondientes a esa tarea.

- **Planeación.** Consiste en la capacidad de crear nuevos programas teleo-reactivos y modificar los existentes por un sistema de planeación automático. A la representación de un programa teleo-reactivo para ser usada por el planificador se le llama TOP (*teleo operator*).

- **Aprendizaje.** Incorpora métodos de aprendizaje de modelos de acciones a partir de ejemplos.

TRAIL representa los programas teleo-reactivos como un árbol al que se le llama árbol TR. La Figura 3.1 muestra la arquitectura de TRAIL.

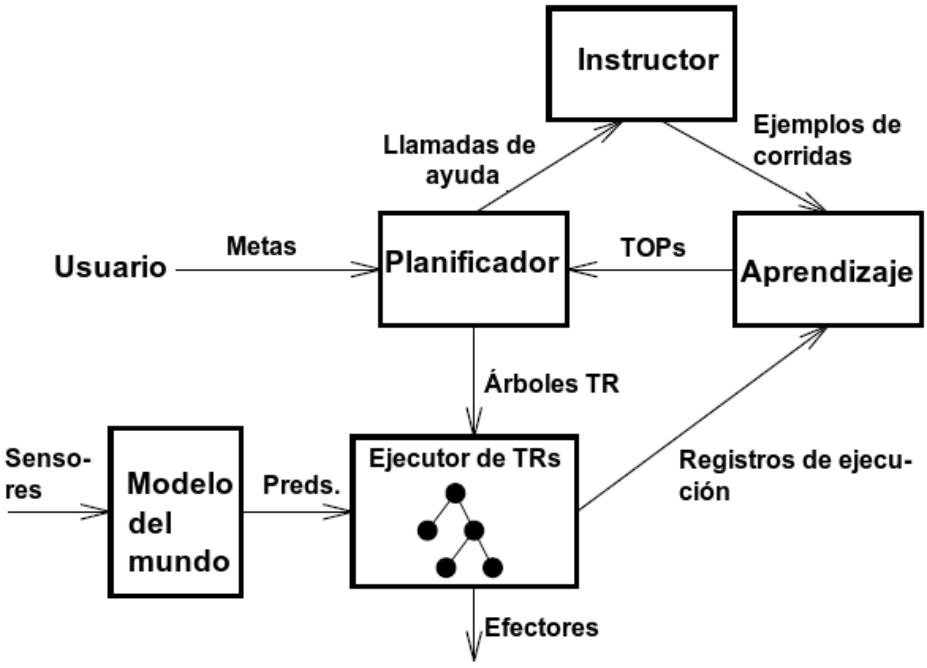


Figura 3.1: Arquitectura de TRAIL

Los componentes de TRAIL son: el modelo del mundo, el planificador, el ejecutor teleo-reactivo, el componente de aprendizaje y el instructor. A continuación se describe la función de cada componente.

- **Modelo del mundo**. En este componente se almacenan los datos de los sensores que son leídos continuamente. A partir de los datos se generan predicados que son la entrada al ejecutor teleo-reactivo.

- **Ejecutor teleo-reactivo**. Su función es determinar cuál nodo en el árbol teleo-reactivo es actualmente el nodo verdadero de mayor prioridad y activar la acción apropiada. Si no existen árboles, el planificador los genera.

- **Planificador**. Recibe las metas en la forma de condiciones y determina cuáles metas se cumplen de acuerdo al ambiente percibido en ese momento. El planificador genera planes en la forma de árboles teleo-reactivos que se espera logren las metas recibidas. Como el planificador depende de un conjunto de modelos de acción producidos por el componente de aprendizaje, cualquier inexactitud en los modelos de acción puede generar árboles incorrectos. La ejecución de los árboles puede llevar al logro de las metas. En el caso de ocurrir alguna falla, TRAIL llamada al componente de aprendizaje o al instructor.

- **Aprendizaje**. El componente de aprendizaje recibe del ejecutor registros de ejecución de acciones y los usa para actualizar su conjunto de modelos de acción (TOPs). El componente corrige inexactitudes que originan fallas en los planes. Los modelos actualizados de acción son usados por el planificador para crear árboles nuevos. El ciclo de aprendizaje supone que TRAIL tiene conocimiento suficiente para generar un plan aproximadamente correcto para la meta que le ha sido dada. Sin embargo, si los modelos de acción son insuficientes para construir un plan, se recurre a explorar el ambiente para lo cual, TRAIL dispone de un instructor externo.

- **Instructor**. Es llamado por TRAIL en cualquier momento que lo necesite, esto es, cuando no tiene información suficiente para que lo guíe a completar su meta actual.

El proceso iterativo de ejecución y planificación se repetirá hasta que TRAIL genera un árbol modificado que sea suficiente para lograr la meta o TRAIL determine que no puede construir un árbol y tenga que recurrir al instructor.

TRAIL fue el primer sistema que utilizó *ILP* para aprendizaje de modelos de acciones. TRAIL utiliza una representación basada en lógica de primer orden y el aprendizaje se hace a partir de trazas que se registran mientras el instructor realiza la tarea. TRAIL es un sistema con un enfoque cercano al trabajo de tesis que aquí se presenta, sin embargo, las diferencias sustanciales son las siguientes:

- TRAIL no se prueba con sensores reales por lo que no se enfrenta al problema del manejo de datos de bajo nivel. En ése tiempo, la tecnología de los robots móviles no estaba lo suficientemente avanzada para probar el sistema en un robot real [Benson, 1996] .
- TRAIL aprende únicamente PTRs básicos, los PTRs compuestos son obtenidos de forma manual y posteriormente re-utilizados.

En [Kochenderfer, 2003] se aprenden programas teleo-reactivos mediante programación genética para resolver el problema de apilamiento de bloques. El sistema aprendió programas con menos reglas que los elaborados por un humano. Las acciones que el sistema maneja son acciones discretas, no duraderas y las variables del dominio de aplicación son contínuas. Una desventaja es que el tiempo de solución es de aproximadamente un par de días por lo que en un dominio de mayor complejidad el tiempo sería aún máyor.

Otra estrategia para aprendizaje de teleo-operadores se muestra en [Broda y Hogger, 2004] y consiste en la construcción sistemática de teleo-operadores basada en una estructura de grafo llamada OPG(*Objective-Perception-Graph*). El grafo representa todas las situaciones en las que el robot puede estar y las acciones que puede realizar. Los arcos representan las acciones y los nodos las situaciones. El grafo se construye a partir de dos conjuntos, el de los estados y el de las posibles observaciones-acciones. Una vez construido el grafo, el objetivo es encontrar una política que permita lograr la meta dada y que a cada situación corresponda una acción. La política es obtenida mediante el método de *discounted rewards* en el que se mide el beneficio de estar en un estado dada una acción. Para un grafo con n situaciones, se obtienen n ecuaciones lineales que se resuelven usando algoritmos estándar. La política obtenida consiste en las reglas que constituyen el conjunto de teleo-operadores necesarios para lograr la meta. El método se probó en el mundo de los bloques y con una representación proposicional por lo que en un dominio más complejo el número de posibles estados puede crecer demasiado. Si el ambiente es modificado, por ejemplo, al agregar otro bloque, el aprendizaje tiene que realizarse nuevamente.

En [Konik y Laird, 2006] se describe un marco para aprendizaje a partir de trazas estructuradas, anotaciones de metas y conocimiento del dominio. Los autores presentan un sistema

de aprendizaje para agentes en el cual el ambiente es representado de forma simbólica por una interfaz y mediante clonación y programación lógica inductiva el sistema aprende los conceptos que después los autores prueban en agentes y evalúan los programas aprendidos comparándolos con agentes cuyo comportamiento fué codificado manualmente. Se planteó el problema como una descomposición del aprendizaje en metas y acciones asumiendo que están representadas jerárquicamente. Las principales diferencias con respecto al trabajo desarrollado en esta tesis son: (i) en el artículo revisado, se supone una jerarquía existente mientras que en nuestro caso, se inicia aprendiendo las acciones primitivas y se va construyendo la jerarquía y (ii) los autores no abordan el problema del uso de sensores reales ya que su dominio de prueba se basa en simulación y sus agentes de prueba utilizan sensores binarios en un nivel muy básico.

En [Zelek y Levine, 1996] se presenta una aplicación del formalismo teleo-reactivo en robots móviles. Un control teleo-reactivo se integra a una arquitectura llamada SPOTT. La extensión consiste en permitir acciones concurrentes mediante operadores adicionales: concurrencia y secuencia. El operador de concurrencia permite que un conjunto de acciones pueda ejecutarse concurrentemente, mientras que el secuencial ordena la ejecución de las acciones de izquierda a derecha. Mediante una interfaz visual se implementaron programas Teleo-Reactivos para construcción de mapas, localización, planeación de trayectorias y ejecución, y fueron probados en un robot Nomad y en una RWI (*Real World Interface*) de forma exitosa. Los problemas que se presentaron se produjeron debido a conflictos en los sensores que originaron un desempeño lento. La arquitectura proporcionó una metodología organizada para programar y controlar un robot móvil. Una limitante de la aplicación es que los programas son codificados manualmente a través de una interfaz.

Resumiendo, el formalismo teleo-reactivo presenta las siguientes ventajas:

- El sensado continuo y la ejecución de acciones de acuerdo a las condiciones del ambiente proporciona al robot la habilidad de reaccionar apropiadamente a modificaciones del ambiente.
- El conjunto de reglas ordenado por prioridades permite que el robot pueda lograr metas dependiendo de la urgencia de cada una.
- El formalismo favorece la integración de diversos componentes como aprendizaje y planeación lo que permite extender las capacidades del robot.

Por otro lado, presenta las siguientes dificultades:

- No es fácil definir los PTRs. Se necesita determinar qué se puede aprender y qué información es más fácil que el usuario proporcione directamente. En el trabajo relacionado, generalmente se programan directamente y los que han sido aprendidos han sido muy limitados.

- Es necesario contar con una plataforma que facilite el sensado continuo del ambiente y lo haga de forma confiable para que el robot pueda responder rápidamente a los cambios.

Contribución. A pesar de que se ha demostrado que el enfoque teleo-reactivo es apropiado para aplicarse en robots, no ha sido ampliamente explorado. En esta tesis se presenta el aprendizaje automático de PTRs y su aplicación en robots móviles. Uno de los principales problemas para obtener una representación que facilite el proceso de los algoritmos de aprendizaje es la gran cantidad de datos que generan los sensores del robot. Para solucionarlo se transforman los datos en una representación compacta gracias a un proceso de identificación de marcas naturales del ambiente. A esta información se le agrega conocimiento del dominio adicional y mediante un sistema de programación lógica inductiva se aprenden reglas de alto nivel, que son los PTRs para el control del robot.

3.4 Aprendizaje relacional

En robótica, el aprendizaje relacional no es una técnica muy difundida. Uno de los principales problemas es la gran cantidad de datos producida por los sensores lo cual dificulta el uso de estos algoritmos. Sin embargo, recientemente el interés por el enfoque relacional se ha renovado y han empezado a surgir trabajos en robótica que buscan aprovechar las ventajas que este tipo de aprendizaje proporciona.

Uno de los primeros trabajos en robótica en el que se ha usado programación lógica inductiva para aprender conceptos a partir de datos de sensores está descrito en [Klingspor et al., 1996]. Los conceptos aprendidos se ubican en un ambiente de oficinas y son del tipo *frente_a_puerta, cruzando_puerta*. Considerando que los ambientes de oficina poseen características similares entre sí, los conceptos aprendidos se basaron en las relaciones entre sus elementos (paredes, puertas) tomando como referencia las diferencias de distancias medidas por los sensores, en este caso, sonares. Pasar por una puerta representaría percibir primero el marco con los sensores frontales, luego por los sensores laterales y posteriormente con los sensores traseros. Los conceptos se caracterizaron por las relaciones observadas entre

intervalos de tiempo y relaciones entre medidas de sensores. La idea fundamental era caracterizar un concepto, por ejemplo: *entrar por una puerta* y distinguirlo de otro como *avanzar por un pasillo* con base en las mediciones de los sensores y los movimientos del robot. Como los conceptos se definieron en forma de relaciones se utilizaron algoritmos de programación lógica inductiva. El proceso general de aprendizaje fue el siguiente:

- Se obtuvieron trazas de datos adquiridos por un robot móvil moviéndose en diversas habitaciones. Los datos correspondían a trayectorias completas.
- Se definieron los predicados que se usarían para describir los conceptos.
- Se aplicó un algoritmo de aprendizaje que restringía el espacio de hipótesis obteniendo un conjunto de cláusulas que definían un concepto.
- Los resultados se evaluaron aplicando los conceptos aprendidos a datos sensoriales adquiridos en otras trayectorias del robot.

Uno de los problemas de los algoritmos de programación lógica inductiva es que no manejan fácilmente conjuntos grandes de datos por lo que se dividió el proceso de aprendizaje en cinco niveles de abstracción:

1. **Aprendizaje de atributos básicos**, consiste en aprender las tendencias en las lecturas de los sensores: `incremento`, `decremento`, `estabilidad`.
2. **Aprendizaje de atributos de cada sensor**, son conceptos que expresan la percepción de los sensores en cierto momento, por ejemplo, *frente-a-la puerta*, *frente-a-una-pared*.
3. **Aprendizaje de configuraciones de grupos de sensores**, en el que se describen configuraciones de elementos durante un cierto movimiento: `cóncava`, `convexa`, `línea`, `salto`.
4. **Integración de atributos de percepción y acción**, donde algunos de los predicados son: *detenerse*, *avanzar*, *movimiento-paralelo*, *movimiento-diagonal*, *rotar*, *acercarse*, *alejarse*.
5. **Conceptos de alto nivel** como: *moverse-cerca-de-la-pared*, *girar-frente-a-la-pared*, *moverse-paralelamente-en-la-esquina*, *pasar-frente-a-la-puerta*, *pasar-a-traves-de-la-puerta* y *girar-frente-a-la-puerta*.

En la evaluación se observaron mejores resultados en la clasificación de situaciones conforme el nivel de abstracción se incrementaba. El porcentaje de exactitud de clasificación de las reglas obtenidas para el nivel más alto fue de aproximadamente 70 %.

Aunque en esta tesis también se aprenden conceptos de alto nivel, el enfoque es muy diferente al empleado en el artículo. Las principales limitaciones observadas son las siguientes:

- El objetivo primordial del artículo es el aprendizaje de conceptos y aunque los niveles 4 y 5 de abstracción ya integran acciones no llegan a ser probadas en tiempo de ejecución. En esta tesis los conceptos se aprenden y se usan para aprender PTRs que son probados en simulación y en un robot real.

- La información sensorial se utiliza directamente en el aprendizaje por lo que para tratar con el problema de la gran cantidad de datos los autores diseñaron un lenguaje de representación en cinco niveles en los que cada concepto está definido en términos de la abstracción del nivel inferior. En nuestro caso, usamos un proceso de identificación de marcas naturales que proporciona solamente la información relevante para usarla posteriormente en una representación relacional.

- El manejo de los cinco niveles de representación no facilita la programación. Para cada nivel, se obtienen distintos conjuntos de ejemplos que son etiquetados manualmente. Por cada nivel existe una fase de aprendizaje con el algoritmo de *ILP*, lo que equivale a tener conjuntos de reglas distintas para cada uno. En esta tesis el aprendizaje se realiza en sólo dos fases: aprendizaje de PTRs básicos y aprendizaje de PTRs compuestos utilizando la misma traza para ambos. El aprendizaje puede incluir conceptos del ambiente como condiciones de los PTRs. El pre-procesamiento que se requiere se realiza automáticamente por lo que nuestra salida son directamente los predicados para identificar conceptos o los PTRs.

En [Cocora et al., 2006] se aprenden planes de navegación usando clonación. A partir de secuencias de lugares atravesados por el robot al ir hacia un punto determinado se aprendieron árboles de decisión relacionales. Estos árboles se usaron para guiar al robot en tareas iguales o parecidas re-utilizando los planes para alcanzar la meta aun en ambientes desconocidos. El mapa del ambiente es etiquetado con los nombres de lugares (*e.g.*, $\texttt{pasillo}_1$, $\texttt{cuarto}_2$) por lo que los planes aprendidos indican los lugares por los que el robot debe pasar para llegar a un punto meta. La limitante es que los planes no consideran eventos inesperados, como por ejemplo obstáculos móviles.

Contribución. En esta tesis se desarrolló una metodología de aprendizaje relacional para aprender reglas de control (PTRs) para robots móviles. Uno de los aspectos fundamentales son los niveles de abstracción utilizados, pues a partir de datos crudos de los sensores se obtuvo una representación de alto nivel mediante un proceso de identificación de marcas naturales y posteriormente una transformación a secuencias de PTRs básicos.

3.5 Aprendizaje por refuerzo

Aunque el aprendizaje por refuerzo no se aplica en esta tesis, en la literatura se encuentra frecuentemente relacionado a las técnicas que usamos. Aprendizaje por refuerzo [Sutton y Barto, 1998] aborda el problema de aprender estrategias de control. El aprendizaje se basa en asignar refuerzos para cada transición estado-acción. La meta es aprender una política que maximice la recompensa total que se recibirá desde cualquier estado inicial.

Aprendizaje por refuerzo consiste en aprender cómo mapear situaciones a acciones para maximizar el refuezo. El aprendiz desconoce qué acciones tomar y tiene que descubrir cuáles le dan mayor refuerzo. Las acciones pueden afectar no sólo el refuerzo inmediato sino la siguiente situación por lo tanto, los subsecuentes refuerzos. La búsqueda por prueba y error y los refuerzos retardados son los aspectos más importantes del aprendizaje por refuerzo.

Una tarea típica de aprendizaje por refuerzo se formula como sigue: un conjunto de posibles estados S, un conjunto de posibles acciones A, una función de transición $\delta : S \times A \rightarrow S$, una función de recompensa $r : S \times A \rightarrow R$. En cada punto en el tiempo, el agente puede estar en uno de los estados s_t de S y seleccionar una acción de acuerdo a su política π. Ejecutar una acción en un estado llevará al agente a un nuevo estado. El agente recibe una recompensa. Se asume que el agente no conoce el efecto de las acciones. El objetivo del aprendizaje es encontrar una política que maximice la suma descontada de las recompensas. La política óptima siempre selecciona la acción que maximiza la suma de la recompensa inmediata y el valor del estado sucesor inmediato.

En aprendizaje por refuerzo, la función de utilidad se representa comúnmente en forma tabular con un valor de salida para uno de entrada. Para nuestro problema, esta representación no es útil pues se tiene un espacio de estados muy grande y se necesita un manejo de acciones continuas. Recientemente se han hecho extensiones como: aprendizaje de funciones, uso de acciones continuas y representaciones relacionales.

Aprendizaje por refuerzo relacional (RRL). En el aprendizaje refuerzo relacional [Džeroski et al., 2001] los estados, acciones y políticas se representan en lógica de primer

orden. Para generalizar se usan algoritmos de programación lógica inductiva (*ILP*). Es posible incluir conocimiento del dominio.

RRL combina el algoritmo Q-learning [Sutton y Barto, 1998] clásico con la selección estocástica de acciones y un algoritmo de *ILP*. En vez de tener una tabla explícita para la función-Q, se aprende una representación en la forma de un árbol lógico llamado árbol-Q. A partir de cada estado meta encontrado se genera un ejemplo para cada acción, representados por un conjunto de hechos. Los ejemplos son la entrada a un algoritmo que induce un árbol relacional en el cual las predicciones corresponden a los valores-Q. El árbol-Q codifica la política óptima asignando un valor Q a todos los posibles pares estado-acción. Para representarla de forma más simple, se usa la función-P que en vez de asignar valores reales a los pares estado-acción, decide si el par estado-acción es óptimo o no. En [Driessens y Dzeroski, 2004] se muestra que introducir ejemplos "buenos" en diversas fases del aprendizaje mejoran los resultados. Al proporcionar ejemplos o trazas de la tarea que se quiere aprender el enfoque se parece a clonación, sin embargo los autores argumentan que en clonación los ejemplos tienen que ser "perfectos" y en RRL no es así. Esta aseveración no es totalmente cierta ya que existen mecanismos para manejo de ruido empleados por algoritmos de *ILP* que permiten manejar ejemplos imperfectos como los que suelen generarse al guiar a un robot.

La Tabla 3.1 muestra las diferencias entre el aprendizaje por refuerzo relacional (RRL) y el aprendizaje de PTRs usando clonación que se presenta en esta tesis. Para el aprendizaje de PTRs se dan como entrada ejemplos positivos y negativos. En RRL no se proporcionan ejemplos, la idea es que durante los episodios se almacenen los pares estado-acción y se utilizan como entrada para inducir el árbol de valores-Q. Los tiempos de aprendizaje son largos y si los estados visitados son insuficientes o no convenientes, por ejemplo que visite los mismos o muy parecidos, no se aprende la tarea. Los experimentos reportados en [Džeroski et al., 2001] y muestran que es necesario proporcionar ejemplos buenos para mejorar los resultados. Los ejemplos que se generan al visitar los estados durante el proceso de aprendizaje no proporcionaron la información necesaria para lograr aprender la tarea. En el caso de clonación, al proporcionar ejemplos positivos y negativos como parte del aprendizaje no se requieren largos tiempos de inducción. Por otro lado, El formalismo teleo-reactivo de los PTRs permite aplicar acciones continuas y manejar metas/sub-metas dentro de la misma representación.

Aunque es una ventaja el hecho de que en RRL el proceso de aprendizaje sea más autónomo, el costo es que depende de los ejemplos que genere que no necesariamente son buenos o suficientes para lograr aprender una política general. Ésto origina que los tiempos sean largos y en ocasiones que la tarea no se aprenda. RRL tiene que utilizar ejemplos para poder

	RRL	PTRs/clonación
Entrada		Ejemplos positivos y negativos
Elementos de formulación del problema	Estados,acciones, conocimiento del dominio episodios,recompensas valores de refuerzo	Estados,acciones, conocimiento del dominio
Proceso de inducción	Árbol relacional (análogo a C4.5) pueden usarse otros algoritmos	ALEPH pueden usarse otros algoritmos
Tiempo de aprendizaje	Horas (mundo de los bloques)	Minutos (evasión/robot)
Salida	Programas lógicos	Programas lógicos
Ventajas	Proceso de aprendizaje más autónomo.	– Menor tiempo de aprendizaje. – El formalismo de PTRs facilita el uso de acciones continuas. – Manejo implícito de metas en la representación.
Desventajas	– Si en el proceso de aprendizaje no se generan ejemplos buenos es posible que no aprenda la tarea. – Mayor tiempo de aprendizaje.	

Tabla 3.1: Comparación entre RLL y PTRs/clonación

mejorar sus resultados lo que le quita la ventaja de autonomía ante clonación. El formalismo teleo-reactivo da a los PTRs manejo de metas y de acciones continuas que aunque RRL puede extender, requeriría agregar mecanismos para lograrlo.

En [Ryan y Pendrith, 1998] se presenta un sistema que combina planeación teleo-reactiva y aprendizaje por refuerzo. El objetivo es acelerar el aprendizaje representando comportamientos en forma de teleo-operadores o PTRs. El esquema de aprendizaje por refuerzo selecciona el PTR a ejecutar. Si tiene éxito, se le da una recompensa o penalización. El proceso se repite hasta que se logra la meta. Se hicieron experimentos en un ambiente simulado de un ambiente de habitaciones en el que un robot tenía que ir de una posición origen a una posición meta. Se comparó el comportamiento de la arquitectura RL-TOPs con un sistema monolítico el cual cubre todo el espacio. Al sistema de RL se le dieron ocho PTRs que correspondían al movimiento de un cuarto a otro adyacente. En las etapas iniciales del aprendizaje, en el sistema monolítico el agente pasó mucho tiempo explorando el mundo sin aprender nada. Sin

embargo, al utilizar un sistema modular, los espacios de aplicación para los comportamientos se redujeron por lo que en una exploración aleatoria se obtuvieron recompensas útiles en poco tiempo. El sistema logra aprender más rápido que con un enfoque tradicional de aprendizaje por refuerzo, sin embargo, los PTRs tienen que ser dados por el usuario.

En [Morales y Sammut, 2004] se describe cómo aprender a controlar un avión usando clonación y aprendizaje por refuerzo. Se mostró cómo con una política aprendida es posible que un avión vuele para cumplir diferentes misiones como por ejemplo, recorrer un circuito.

3.6 Aprendizaje a partir de secuencias

La segunda parte de la tesis trata sobre la manera en la que una vez aprendido un conjunto de habilidades, éstas se combinen de forma automática para aprender una habilidad nueva. En la literatura, este problema se encuentra bajo distintos nombres, por ejemplo, aprendizaje de jerarquías y descomposición de tareas. La identificación automática de acciones compuestas es importante porque libera al programador de realizar el trabajo de forma manual.

Una tarea puede verse como una secuencia de acciones de naturaleza jerárquica. En una secuencia pueden encontrarse acciones simples o básicas y acciones compuestas, es decir, formadas por dos o más acciones básicas. Una de las debilidades encontradas en los sistemas de aprendizaje es que las jerarquías deben darse de forma explícita por el programador, por ejemplo, mediante plantillas (*templates*) que contienen campos para definir cuál comportamiento es jerárquico y las acciones que lo componen [Nicolescu y Mataric, 2002].

Una técnica que muestra la descomposición de tareas, es el aprendizaje por refuerzo jerárquico [Ryan y Reid, 2000] para el aprendizaje de vuelo. Se proporcionan al sistema diversos PTRs para aprender comportamientos usando aprendizaje por refuerzo. Los PTRs son combinadas por un planeador para resolver tareas más complejas. Para representar una jerarquía se definen comportamientos a distintos niveles de granularidad. Por ejemplo, los comportamientos del nivel más alto se descomponen en comportamientos más simples y todos cubren el espacio de estados completo. El aprendizaje realizado con los comportamientos del segundo nivel de descomposición mostró mejores resultados que con el aprendizaje no-jerárquico. Sin embargo, al combinar los comportamientos de diversos niveles, los resultados no fueron mejores. El tiempo de aprendizaje es muy largo (aproximadamente 6 horas de vuelo simulado). Para cada nivel de jerarquía los comportamientos son dados por el usuario, mientras que en nuestro enfoque, los comportamientos a distintos niveles son aprendidos.

Descubrir la estructura jerárquica de una secuencia es un problema importante en diversas áreas. El procesamiento de lenguaje natural, compresión de textos, análisis de música y de ADN son dominios que explotan la información proporcionada por la estructura de sus secuencias [de la Higuera, 2000]. Las técnicas de inducción de gramáticas extraen este tipo de información pero generalmente dependen de tener las secuencias segmentadas, donde los segmentos son ejemplos de la estructura que se quiere aprender. En este contexto, los elementos de las secuencias son símbolos discretos como letras, notas musicales o distintas unidades lingüísticas que representan datos válidos de entrada.

La inducción de gramáticas ha sido comúnmente estudiada en el contexto de procesamiento de lenguaje natural. Las secuencias de palabras son transformadas a categorías sintácticas de las cuales las reglas gramaticales son inducidas. En otros dominios lo que tenemos es una secuencia de acciones capaces de ejecutar una tarea, y lo que queremos aprender es la estructura jerárquica capaz de reconocer y reproducir secuencias para realizar la tarea.

La inducción de gramáticas consiste de manera general en encontrar, a partir de datos, una gramática o autómata que sea capaz de producir un conjunto dado de secuencias, árboles, términos o grafos [de la Higuera y Oats, 2006]. La estructura jerárquica de una secuencia puede proporcionar conocimiento al cual se le puede dar diversos usos dependiendo del dominio de aplicación. En esta tesis, la inducción de gramáticas tiene por objetivo aprender programas a partir de ejemplos que se representan como secuencias de acciones. Las reglas de la gramática corresponden a predicados que se utilizan para controlar a un robot.

Una técnica de inducción de gramáticas estrechamente relacionada a nuestro trabajo es la de SEQUITUR [Nevill-Manning y Witten, 1997], un algoritmo que infiere una estructura jerárquica a partir de secuencias de símbolos discretos. Cada vez que SEQUITUR encuentra una cadena repetida, crea una regla y la subsecuencia repetida es reemplazada por un símbolo no-terminal. Sin embargo, SEQUITUR: (i) maneja sólo símbolos constantes, (ii) está basado en bigramas (conjuntos de dos símbolos consecutivos), y consecuentemente, las reglas aprendidas se forman por pares de elementos, y (iii) al aplicarse sólo a constantes no incorpora ningún tipo de generalización. Las dos principales aplicaciones se realizan en texto: compresión y búsqueda de la estructura jerárquica. En esta tesis se presenta un algoritmo relacional para inducción de gramáticas que permite generalizar entre las reglas aprendidas.

EMILE [Adriaans et al., 2000] y ABL [van Zaanen, 2000] son algoritmos basados en lógica de primer orden que infieren la estructura gramatical de un lenguaje. EMILE utiliza *clustering* y trata de encontrar clases de expresiones que pueda agrupar y comparar en otros contextos. ABL [van Zaanen, 2000] se basa en la comparación y la alineación de oraciones planas. Utiliza

las diferencias entre las oraciones para encontrar posibles palabras o conjuntos de palabras (constituyentes) que puedan ser intercambiadas para seleccionar los más probables. La salida es una versión estructurada del corpus. Ambos son algoritmos de aplicación específica por lo que no son fácilmente extendibles a otros dominios.

Un algoritmo que aprende programas lógicos es GIFT [Bernard y de la Higuera, 1999]. Dado un conjunto de ejemplos positivos y negativos, un experto construye un conjunto de reglas para analizar sintácticamente los ejemplos. Estas reglas son usadas para asociar un árbol a cada ejemplo. Él árbol se transforma en un término que se usa como entrada de un algoritmo de inferencia gramatical que da como salida un autómata determinístico. El autómata se combina con conocimiento del dominio para producir un programa lógico para el concepto descrito por los ejemplos. Aunque GIFT aprende programas lógicos, requiere de un conjunto inicial de reglas dado por un experto. En contraste, el algoritmo que se presenta en esta tesis aprende gramáticas relacionales a partir de secuencias de acciones y no requiere reglas iniciales. Las gramáticas resultantes son programas lógicos que pueden reproducir la tarea descrita por las secuencias. La Tabla 3.2 muestra una tabla comparativa de los algoritmos descritos. La columna "Entrada" indica el tipo de información requerida por el algoritmo. La "estructura explícita" indica si la salida del algoritmo muestra claramente la estructura jerárquica que se aprende. La siguiente columna indica si el algoritmo se basa en lógica de predicados. La columna "generalización" se refiere a la capacidad del algoritmo de generalizar las reglas aprendidas. La columna "aprende programas lógicos" dice si la salida del algoritmo consiste en programas lógicos.

En nuestro caso, los elementos de las secuencias de entrada pueden ser símbolos o términos que representan acciones o PTRs. Cada secuencia representa en conjunto una tarea o movimiento, y lo que se busca es aprender la estructura y generar programas lógicos que

Algoritmo	Entrada	Estructura explícita	Manejo de predicados	Generalización	Aprende programas lógicos
SEQUITUR	Secuencias	Sí	No	No	No
ABL	Oraciones	Sí	Sí	No	No
EMILE	Oraciones	No	Sí	No	No
GIFT	Ejemplos, reglas, árbol para asociar ejemplos a las reglas	Sí	Sí	Sí	Sí

Tabla 3.2: Comparación de algoritmos para inducción de gramáticas

reproduzcan las secuencias y también permitan clasificarlas. Por esta razón, fue necesario implementar un algoritmo que construyera programas lógicos a partir de secuencias. En el Capítulo 5 se describe el algoritmo detalladamente.

El aprendizaje a partir de secuencias ha sido utilizado en robótica para el aprendizaje de habilidades. En [Amit y Mataric, 2002] se usa este enfoque en el contexto de aprendizaje por imitación para aprender movimientos (*e.g.*, ejercicios aeróbicos) mostrados por un instructor y posteriormente para reconocerlos se usaron modelos ocultos de Markov (HMM). Esta representación no es fácil de interpretar y no captura la estructura jerárquica de las secuencias.

Otro enfoque en robótica consiste en aprender comportamientos de acciones primitivas [Boucher y Dominey, 2006]. Las acciones se expresan como predicados de primer orden. El proceso consta de dos fases, en la primera, el usuario guía al robot para ejecutar el comportamiento y la secuencia de acciones generada es agregada a un repertorio de secuencias. En la segunda fase, después de distintos episodios, el sistema generaliza la secuencia, representada como grafos. Después del aprendizaje, el robot es capaz de realizar el comportamiento. Algunos ejemplos de los comportamientos aprendidos son "acercarse", "alinearse" y "girar". No se aprenden comportamientos compuestos.

El reconocimiento de ademanes es una habilidad importante para interactuar con las personas. Los modelos ocultos de Markov (HMM) y las redes neuronales son las técnicas estándard para reconocimiento de ademanes. Sin embargo, la mayoría de las técnicas ha enfatizado la mejora del reconocimiento sin considerar la claridad de la representación. En [Avilés-Arriaga et al., 2006] se presenta un enfoque basado en HMMs en el cual se incluyen atributos de movimiento y postura para mejorar la clasificación. En [Westeyn et al., 2003] se muestra la definición de gramáticas para el reconocimiento en una secuencia. Sin embargo, las gramáticas son usadas como una técnica auxiliar y no son aprendidas.

En [Ogale et al., 2007] se construye una gramática probabilista libre de contexto para reconocer posturas humanas a partir de secuencias de video. La limitante es que el sistema describe acciones usando posiciones estáticas. En la presente tesis se aprenden gramáticas para un conjunto de ademanes dinámicos que se usan para reconocer nuevas secuencias. A partir de cada gramática, la estructura del ademán puede analizarse. En [Tonaru et al., 2009] se representan actividades como secuencias simbólicas de acciones. Ejemplos de las actividades son: tomar café, quitarse los audífonos, rascarse la cabeza. Las actividades se extraen basándose en la frecuencia, de forma parecida a la que FOSeq aprende las reglas. Sin embargo, en [Tonaru et al., 2009] se hace énfasis en el aprendizaje de la secuencia y no en la

gramática que la produce.

Un algoritmo para aprendizaje relacional de conceptos basado en grafos es SubdueCL [González et al.,]. SubdueCL encuentra subestructuras que aparecen repetitivamente en el grafo que representa una base de datos. En este sentido, la idea básica es parecida a la del enfoque de aprendizaje de gramáticas que se presenta en esta tesis. Subdue muestra resultados competitivos en comparación con sistemas de *ILP* como FOIL y Progol. Sin embargo, con SubdueCL aun no es posible aprender rangos de valores lo que para nuestro caso es necesario. En esta tesis el aprendizaje de rangos se hace en la fase de aprendizaje de PTRs básicos utilizando ALEPH.

La inferencia de la estructura jerárquica en estos dominios proporciona información significativa sobre cómo ejecutar la tarea o movimiento y para reconocimiento. Es deseable obtener una gramática expresiva que nos permita interpretar claramente la estructura de la tarea o movimiento. Sin embargo, una secuencia de movimientos es generalmente información de bajo nivel obtenida de los sensores lo que dificulta al proceso de aprendizaje generar una salida fácil de entender. En nuestro problema, los ejemplos de entrada son secuencias cuyos elementos son acciones. El objetivo es aprender reglas para reconocimiento de secuencias que reproduzcan la tarea que se está aprendiendo. No se cuenta con información inicial para segmentar o conocer una estructura gramatical base para aprender.

Resumiendo, la inducción de gramáticas proporciona las siguientes ventajas:

- Extrae la estructura jerárquica de una secuencia.
- Permite reconocer secuencias por lo que puede utilizarse como clasificador.
- Es posible construir programas a partir de las reglas de la gramática además de obtener reglas fácilmente interpretables.

Sin embargo, el proceso puede ser costoso computacionalmente para secuencias muy grandes.

Contribución. En esta tesis se presenta FOSeq, un algoritmo que aprende gramáticas a partir de secuencias de acciones. Las secuencias son dadas como trazas de bajo nivel de datos de sensores que son transformados en una representación de alto nivel. FOSeq induce una gramática para cada secuencia y realiza un proceso de generalización sobre la mejor gramática para cubrir la mayoría de las secuencias. FOSeq es capaz de aprender gramáticas a partir de secuencias de símbolos o de secuencias de predicados de primer orden. Las gramáticas

inducidas por FOSeq pueden ser usadas para ejecutar una tarea particular o para clasificar nuevas secuencias.

3.7 Resumen

En este capítulo se presentó una revisión del estado del arte en la que se describen diversos enfoques para resolver los problemas que se abordan en esta tesis. Se identifican los siguientes problemas:

- La clonación en robótica se ha usado con un enfoque proposicional que no es generalizable y aplicable a otros dominios similares. También dificulta la incorporación de conocimiento del dominio y el uso combinado de los modelos aprendidos.
- La mayoría de los trabajos relacionados con PTRs se realizan en simulación por lo que no se cubre el problema relacionado al manejo de datos generados por sensores reales. En pocos trabajos el aprendizaje es automático y los existentes son limitados.
- En cuanto al aprendizaje relacional, uno de los principales problemas en robótica es obtener una representación que facilite el manejo de gran cantidad de datos.
- El aprendizaje de PTRs usando clonación se realiza en menores tiempos que los reportados en aprendizaje por refuerzo relacional. El uso de PTRs proporciona un formalismo para manejo de acciones continuas y metas que aunque en aprendizaje por refuerzo puede extenderse, involucra integrar otros mecanismos.
- El aprendizaje de gramáticas se dirige principalmente al área de procesamiento de lenguaje por lo que los algoritmos existentes son muy específicos. Los trabajos existentes sobre aplicación de gramáticas para secuencias de acciones son escasos. Las gramáticas se construyen de forma manual y se usan como apoyo más que como técnica principal. No se encontraron trabajos en los que se aprendan programas de control a partir de gramáticas.

En el siguiente capítulo se describe el aprendizaje de PTRs básicos utilizando clonación e *ILP* para tareas de navegación en un robot móvil.

4

Aprendizaje de PTRs Básicos

En este capítulo se describe el proceso de aprendizaje de PTRs básicos. En esta fase se definen las sub-tareas de navegación para las cuales se van a aprender PTRs. El usuario guía al robot para realizar la sub-tarea y se obtienen trazas de su ejecución. Las trazas están formadas por un conjunto de lecturas crudas de los sensores el cual es transformado en una representación compacta, de alto nivel de abstracción, basados en marcas naturales del ambiente. Esta información, junto con conocimiento del dominio, se entrega a un sistema de *ILP* para inferir reglas de alto nivel, que al ser combinadas permitirán posterioremente resolver tareas más difíciles. Se presenta el proceso de aprendizaje de PTRs para diversas sub-tareas de navegación: ***deambular***, ***ir a un punto***, ***seguir*** a un objeto móvil y ***salir de trampas***.

4.1 Introducción

Para una persona es muy natural ir de un lugar a otro aún sin conocer específicamente un mapa del ambiente. Si se le presentan obstáculos, los rodea aparentemente sin pensar cómo hacerlo, retomando su camino hacia su destino. Sin embargo, programar un robot para que tenga ésas habilidades es una tarea laboriosa y complicada. Como vimos en la sección 2.6, los Programas Teleo-Reactivos (PTRs) constituyen un marco efectivo para el control de robots con diversas metas en ambientes dinámicos. Sin embargo, para su aprendizaje es necesario contar con una representación que facilite la aplicación de algoritmos y su uso en diversas tareas. La idea es aprender PTRs que sean simples y expresivos con el fin de que se incorporen a arquitecturas de robots de servicio de forma transparente al programador.

Se ha seleccionado la navegación de un robot móvil para la aplicación de la plataforma de aprendizaje propuesta.

> Navegación es el proceso de determinar y mantener un rumbo o trayectoria hacia una ubicación meta [Franz y Mallot, 2000].

De acuerdo a esta definición, las capacidades mínimas para navegar son desplazarse en el espacio y determinar si se ha llegado a la meta o no. Sin embargo, la navegación de un robot móvil involucra un conjunto de habilidades que puede llegar a ser muy amplio. Las habilidades para navegación pueden ser: capacidad de determinar su propia posición en un mapa, planear una trayectoria, evadir obstáculos, construcción de mapas y su interpretación entre otras.

En esta tesis, la aplicación de navegación se enfoca en la generación de una representación del ambiente, el aprendizaje de habilidades para evasión, orientación, aprendizaje de conceptos para identificar zonas específicas y manejo de metas en ambientes dinámicos. El proceso incluye la construcción de mapas [Jáquez, 2005] y localización [Hernández y Morales, 2006] del robot en el ambiente usando módulos ya existentes.

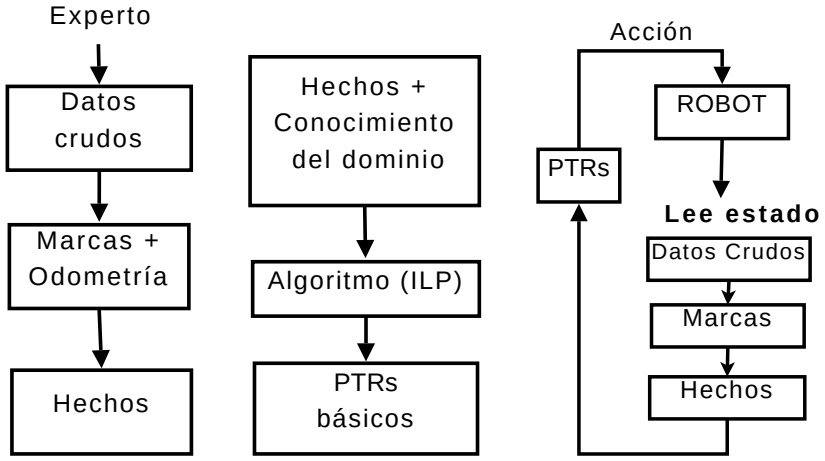
El objetivo de la navegación en esta tesis es aprender a desplazarse de un punto A a un punto B sin chocar con muebles, paredes o personas. El robot necesita tener habilidades que le permitan detectar obstáculos, evadirlos y evitar caer en trampas o lugares de difícil acceso o salida. Se supone que el robot opera en ambientes domésticos y que para el entrenamiento se cuenta en todo momento con un "instructor".

Este capítulo presenta el proceso de aprendizaje de PTRs básicos de navegación, es decir, aquellos PTRs cuyas acciones son instrucciones de bajo nivel (*e.g.*, `girar-izquierda`, `detenerse`) y que posteriormente puedan ser usados como acciones para el aprendizaje de PTRs jerárquicos, que son aquellos que en su conjunto de acciones incluyen a otros PTRs.

4.2 Esquema de aprendizaje

En esta sección se describe de forma general cómo un robot aprende PTRs básicos a partir de ejemplos. Una persona entrena al robot guiándolo con el *joystick* para que realice una tarea determinada. La persona, a la que le llamamos "experto" generalmente practica previamente la ejecución de la tarea guiando al robot antes de registrar los ejemplos. La idea es que los ejemplos obtenidos sean buenos aunque el algoritmo de *ILP* tolera errores o inconsistencias. Durante la ejecución de la tarea, las lecturas de los sensores son registradas.

A partir de estas lecturas se construyen los ejemplos que servirán para que el robot aprenda a realizar la tarea en ambientes dinámicos similares a aquéllos en los que se entrenó. El proceso completo consta de las siguientes fases (Figura 4.1):



(a) Adquisición y transformación de trazas

(b) Aprendizaje de PTRs básicos

(c) Prueba

Figura 4.1: Fases del aprendizaje de PTRs: (a) Una persona (el experto) entrena al robot guiándolo con el *joystick* para que realice una tarea determinada. Para cada tarea se obtienen trazas de información de bajo nivel que son tranformadas en marcas naturales. Las lecturas de odometría son agregadas a la información de las trazas. La transformación de datos crudos en información de marcas permite construir una representación de alto nivel consistente en hechos de Prolog que en (b) se les agrega conocimiento del dominio adicional, como por ejemplo, conceptos aprendidos previamente. Los PTRs básicos son inducidos con el sistema de *ILP*, ALEPH y en (c) los PTRs son probados en simulación y posteriormente en el robot real

- **Adquisición y transformación de trazas.** Las trazas de información de bajo nivel obtenidas son transformadas en trazas de alto nivel basadas en marcas naturales. Las lecturas del sensor láser se convierten en marcas naturales del ambiente mediante un proceso de identificación de paredes, esquinas y discontinuidades. El objetivo de este proceso es producir un conjunto pequeño de datos que sean significativos y disminuya la carga, facilitando la representación y el uso de algoritmos de aprendizaje.

- **Aprendizaje de conceptos.** De las trazas de alto nivel obtenidas se aprenden conceptos que describen la zona que el robot percibe (*e.g.*, `zona-frontal`). Los conceptos son aprendidos usando el sistema de *ILP*, ALEPH [Srinivasan-ALEPH-url, 1999]. Estos

conceptos son útiles como conocimiento del dominio para el aprendizaje de PTRs para sub-tareas de navegación.

- **Aprendizaje de PTRs básicos.** Las trazas de alto nivel, junto con conocimiento del dominio adicional son la entrada a ALEPH. Los conceptos de zona aprendidos pueden usarse como conocimiento del dominio. La salida es un conjunto de PTRs que expresan cuándo ejecutar una acción para realizar sub-tareas simples, *e.g.*, ***deambular***.

- **Prueba.** Los PTRs aprendidos se prueban en simulación, para ello, se determina un número de tareas o sub-tareas para que el robot ejecute. Si no se cumple con una precisión mayor o igual al 80 % el aprendizaje se repite. Posteriormente los PTRs son probados en el robot real.

En la siguiente sección se describe cómo a partir de lecturas de datos de los sensores se obtiene una representación del mundo para el aprendizaje.

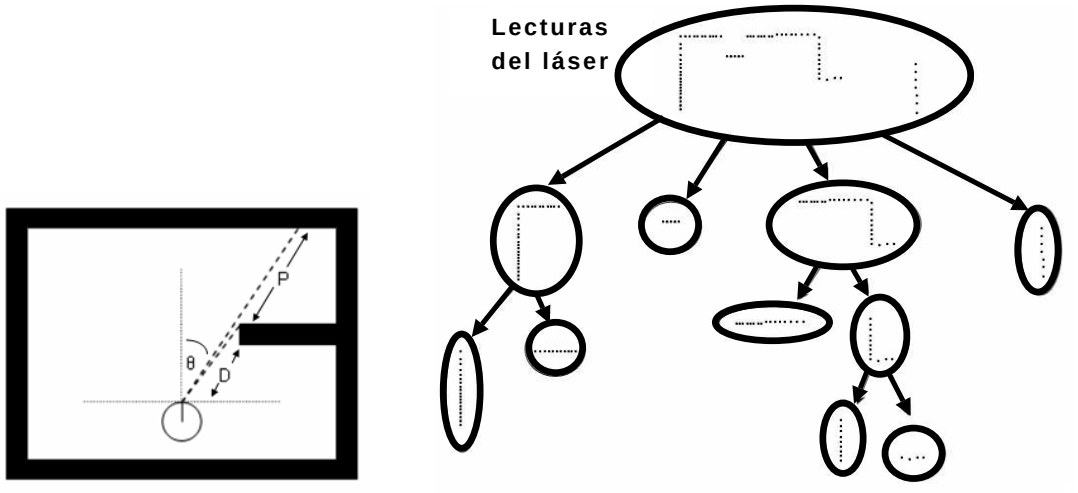
4.3 Representación del ambiente

Cuando una persona se desplaza en su casa o en su oficina es capaz de identificar los objetos a su alrededor tales como mesas, sillas, paredes, puertas. Esta identificación le permite a la persona tomar decisiones, por ejemplo, sobre cómo moverse para rodear una mesa y llegar al extremo opuesto de una habitación. Sin embargo, la capacidad visual de una persona es muy diferente a lo que un sensor como un láser o un sonar pueden percibir del ambiente. Un robot que utiliza este tipo de sensores para desplazarse recibe mediciones de la distancia hacia los objetos que percibe. Esta información cruda no es fácil de manejar pues generalmente se recibe en grandes cantidades y es difícil determinar qué mediciones son significativas y cuáles no son relevantes para la tarea que se quiere aprender. Lo que se desea es que el robot pueda construir una representación de mayor nivel que extraiga información que facilite el uso de algoritmos de aprendizaje. En esta sección se muestra cómo, a partir de lecturas de los sensores, se identifican paredes, esquinas y discontinuidades del ambiente para construir una representación de alto nivel.

Cuando el instructor guía al robot con el *joystick* para que realice una tarea, los datos de los sensores se registran en trazas. El proceso de adquisición de trazas es realizado en simulación utilizando la plataforma *Player/Stage* [Vaughan et al., 2003] con un modelo del robot Pioneer 2, sensor láser y un anillo de 16 sonares. La plataforma de simulación proporciona

el modelo del robot y los modelos de los sensores. El láser realiza lecturas en la parte frontal del robot, cubriendo una zona de 0 a 180 *grados* con un alcance de 8 *metros*. Del anillo de sonares se registran los valores de los 8 sonares de la parte trasera así como las variables de odometría: posición X, posición Y y orientación. Tenemos un total de 191 variables (180 del láser, 8 de los sonares y 3 de posición y orientación), todas de naturaleza continua. Para generar un conjunto reducido de información significativa, el sistema utiliza un proceso de identificación de marcas naturales [Hernández y Morales, 2006]: discontinuidades, esquinas y paredes. Este proceso se describe a continuación.

- **Discontinuidades.** Una discontinuidad se define como una variación significativa en la distancia entre lecturas consecutivas del láser como se muestra en la Figura 4.2(a). Una vez que se ha identificado un conjunto de discontinuidades, el rango completo es dividido en segmentos. Ésto corresponde al primer nivel del árbol de segmentos mostrado en la Figura 4.2(b).



(a) (b)

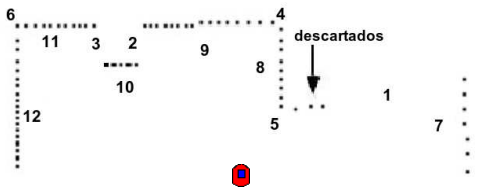
Figura 4.2: (a) Ejemplo de discontinuidad, (b) Árbol de segmentos después de identificar discontinuidades y esquinas

- **Esquinas.** Cada segmento se usa para tratar de identificar esquinas. Para ello se obtiene el punto más lejano en el segmento desde una línea imaginaria pasando por el punto inicial y final del segmento. Si la distancia entre este punto y la línea imaginaria es mayor que un umbral, entonces es considerado esquina. El segmento se divide usando la esquina y el mismo proceso se repite para cada sub-segmento. Como se ve en la

Figura 4.2(b), al final del proceso, la división de segmentos puede verse como un árbol de segmentos cuyas hojas son posibles paredes.

- **Paredes.** Cada hoja del árbol es tomada y se realiza una transformada local de Hough para calcular los parámetros de las líneas correspondientes. Esta transformada es rápida porque existe una sola línea posible en cada segmento.

Una marca natural está representada por una tupla de cuatro atributos: D, θ, A, T, donde D y θ corresponden a la distancia de la marca al robot y la orientación de la marca en relación al robot respectivamente. T es el tipo de marca: i para discontinuidad izquierda, d para discontinuidad derecha, e para esquinas y p para paredes. A es un atributo distintivo cuyo su valor depende del tipo de marca: para discontinuidades A es la profundidad y para paredes A es su longitud. La Figura 4.3 muestra un ejemplo de marcas naturales identificadas aplicando el proceso previo.



(a)

No	D	θ	A	T
12	1.10	89.75	1.43	p (pared)
6	1.84	36.00	0.00	e (esquina)
11	1.50	00.00	0.80	p (pared)
3	1.11	10.00	0.41	d (discontinuidad derecha)
10	1.10	00.00	0.39	p (pared)
2	1.10	-6.00	0.40	i (discontinuidad izquierda)
9	1.49	00.00	1.36	p (pared)
4	2.17	-47.00	0.00	e (esquina)
8	1.59	-89.75	0.85	p (pared)
5	1.71	-68.00	0.00	e (esquina)
1	2.18	-73.00	1.41	i (discontinuidad izquierda)
7	3.46	-89.50	1.06	p (pared)

(b)

Figura 4.3: A partir de las lecturas del láser (a) se identifican las marcas naturales del ambiente. El conjunto de marcas (b) representa: la distancia en metros del robot hacia la marca (D), la orientación del robot con respecto a la marca (θ), el atributo de la marca (A) y su tipo (T)

Trazas con información de marcas. Mientras el robot es guiado, el proceso de identificación de marcas transforma los datos crudos en esquinas, paredes y discontinuidades. Esta información, junto con la de odometría es registrada en trazas en la forma de pares estado-acción (ver Tabla 4.1) donde Xr y Yr son las coordenadas del robot relativas al mapa del ambiente; Thr es la orientación del robot dentro del un rango de 0 a 360 *grados*; D, Th, Att y $Tipo$ son los 4 atributos que describen la marca. El atributo RO indica si hay

un obstáculo en la parte trasera del robot y se extrae de las mediciones del anillo de sonares. La acción es obtenida a partir de la odometría: `avanzar` representa un desplazamiento de al menos 0.3 m/s y `girar-derecha` y `girar-izquierda` representan un desplazamiento de 5 $grados/s$. Como se observa en la Tabla 4.1 cada estado puede tener varias marcas, en el ejemplo mostrado los tres estados tienen 2 marcas.

	Posición			Marcas					
	Xr	Yr	Thr	D	Th	Att	$Tipo$	RO	*Acción*
Estado 1	-6.35	2.80	0.00	5.54	-63.27	0.00	e	n	avanzar
				6.29	-84.06	1.48	p		
Estado 2	-4.10	2.80	1.00	4.95	-90.00	0.43	p	n	girar-derecha
				1.10	83.25	0.25	p		
Estado 3	-7.13	1.61	168.00	5.07	74.34	0.00	e	n	girar-izquierda
				5.53	83.25	0.39	p		

Tabla 4.1: Traza con información de marcas. Se construyen pares estado-acción donde el estado está formado por la posición del robot (*Xr,Yr,Thr*), el conjunto de marcas naturales identificadas *(D, Th, Att, Tipo)* y la información de los sonares (*RO*). La acción (*Acción*) es aquélla ejecutada por el usuario al guiar al robot

Mientras que una traza de datos crudos tiene 191 atributos: 180 del sensor láser, 8 del anillo de sonares, 3 de odometría y el atributo de clase. Después de la transformación la traza de alto nivel tiene en total 9 atributos: 3 atributos de odometría, 1 atributo del sonar, 4 atributos de cada marca que percibe y el atributo de clase, en este caso la acción.

Trazas con ejemplos expresados en lógica de primer orden. Después del proceso de identificación de marcas naturales se tienen ejemplos expresados como atributos asociados a una clase. Sin embargo, se requiere una representación que exprese las relaciones entre los objetos y permita usar los algoritmos de *ILP*. Esta información es transformada en un pequeño conjunto de predicados. Por ejemplo, para la posición del robot en la Figura 4.4, un posible par estado-acción es el siguiente:

$[robot(4.85,\text{-}0.04,248.60), atrás(\texttt{n}), meta(0,0), marca(1.19,\text{-}26.65,3.98,d), marca(2.17,30.67,0.78,\texttt{i}),$
$marca(0.78,\text{-}29.66,0.00,\texttt{e}), marca(0.79,\text{-}30.67,1.85,\texttt{p}), \ldots]$, `detenerse`

donde: (i) $robot(X,Y,Thr)$ representa la posición en X,Y del robot y Thr su orientación, (ii) *atrás(RO)* indica si existe un obstáculo en la parte trasera del robot y es dado por las lecturas del anillo de sonares. Puede tomar el valor `s` ó `n`, (iii) $marca(DL,ThL,Att,Tipo)$

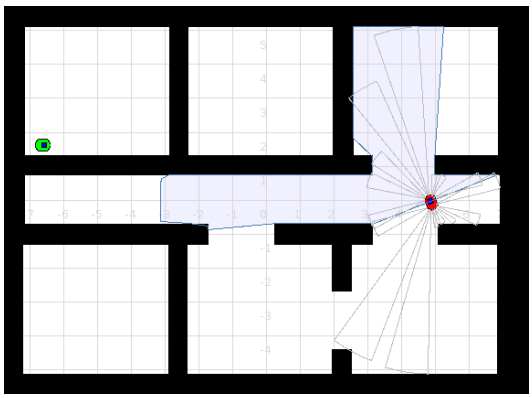


Figura 4.4: Ejemplo de un estado particular del robot mostrando las lecturas del láser

corresponde al conjunto de marcas que el sensor percibe y sus cuatro atributos. El número de marcas que el sensor puede percibir es variable, y (iv) *Acción* es la acción que se ejecutó en este estado, en el ejemplo fue `detenerse`. Los sensores proporcionan una descripción de cada estado del ambiente que es usada como ejemplo o instancia de aprendizaje.

En la siguiente sección se definen los PTRs de navegación que se aprenderán utilizando la representación de marcas.

4.4 Definición de la tarea de navegación y sus sub-tareas

La navegación de un robot móvil en un ambiente doméstico es una tarea que presenta retos ya que es muy difícil conocer de antemano todos los posibles estados del ambiente a los que el robot debe enfrentarse. Navegación es una tarea necesaria para la realización de actividades tales como mensajería y seguimiento, por lo que es deseable que el conjunto de PTRs resultante pueda integrarse fácilmente a las tareas de mayor jerarquía que requieran de esta capacidad sin que el programador tenga que estar pensando en aspectos de bajo nivel. En cada paso de tiempo el robot conoce su posición X, Y, θ en el ambiente en el cual se encuentra pero no conoce el mapa.

Una vez conocida la representación del ambiente, el siguiente paso es determinar la tarea y/o sub-tareas para las cuales se aprenderán PTRs que las realicen y es necesario definir qué elementos se consideran útiles para lograrlo. En la Figura 4.5 se muestra una trayectoria del desplazamiento del robot desde un punto origen hacia un punto destino. En su trayecto el robot tiene que evitar chocar con paredes, con objetos móviles, orientarse contínuamente

hacia la meta y salir de lugares estrechos.

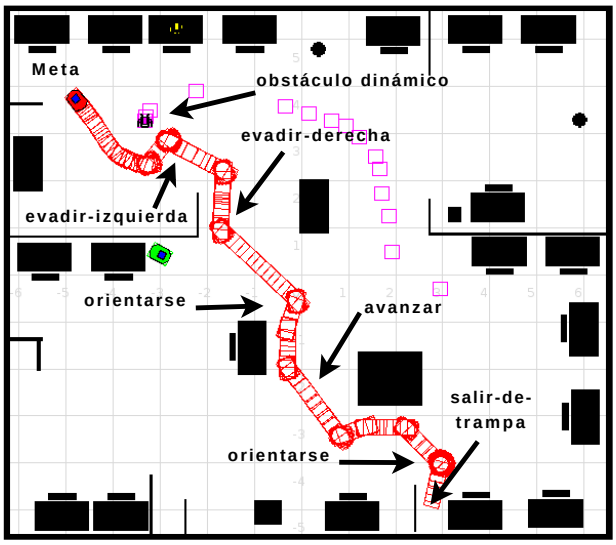


Figura 4.5: Navegación. El robot durante su desplazamiento de un punto a otro tiene que salir de trampas, evitar chocar con paredes y mobiliario, *avanzar* y orientarse hacia la meta

Las sub-tareas de navegación para las que se aprenderán PTRs que las resuelvan son: (i) evasión de obstáculos y avance en espacios libres (***deambular***), (ii) ***orientarse*** hacia el punto meta y (iii) ***salir de trampas***. Como trampa nos referimos a espacios estrechos y cerrados en los que el robot puede entrar pero de los que tiene dificultades para salir. Los PTRs para cada sub-tarea pueden aprenderse y probarse como PTRs independientes.

Para el aprendizaje de los PTRs de navegación puede proporcionarse conocimiento adicional. Existen áreas en el ambiente que pueden describirse como zonas específicas con respecto al robot. Por ejemplo, ¿qué condiciones determinan que hay obstáculos en la *zona-frontal* del robot?, ¿y en la *zona-trasera*?, ¿cómo se describe una zona para orientarse hacia una meta dada?, ¿cómo es una zona de trampa?. Aprender una descripción de este tipo de conceptos es de gran utilidad para el aprendizaje de los PTRs de navegación pues los conceptos pueden darse como conocimiento del dominio. En la siguiente sección se describe cómo aprender este tipo de conceptos.

4.5 Aprendizaje de conceptos para navegación

El ambiente donde el robot navega puede describirse mediante conceptos que representan lo que el robot percibe. Por ejemplo, cuando una persona camina, generalmente observa al

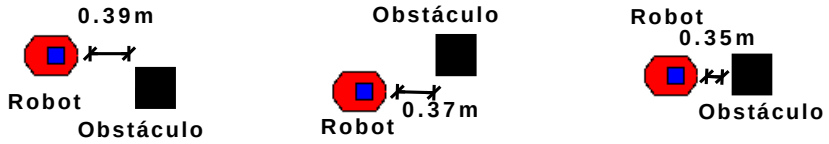
frente y sus movimientos son definidos por lo que va viendo. De manera análoga, para que un robot deambule es importante lo que sus sensores perciben frente a él pues puede tratarse de un obstáculo al que se debe evadir. Considerando ésto, pueden aprenderse conceptos para navegación que son útiles como conocimiento del dominio disponible para aprender PTRs. La idea es que los conceptos que se aprendan sean útiles en todo el dominio, por ejemplo, un obstáculo frente al robot será un concepto usado para cualquier tarea en la que el robot necesite desplazarse. Los conceptos son definidos por el usuario y se realizan con base en conocimiento que éste pueda dar sobre el dominio y que considere que pueda ser útil para la construcción de la hipótesis. Se ejemplifica el proceso de aprendizaje con el concepto *zona-frontal* que describe lo que el robot ve frente a él desde una posición. La descripción incluye la marca más cercana, el valor de distancia considerado como riesgoso y la orientación de la marca en relación al robot.

Para aprender el concepto *zona-frontal* con *ILP* se necesita un conjunto de ejemplos positivos, un conjunto de ejemplos negativos, y posible conocimiento del dominio. El objetivo es encontrar una hipótesis que cubra los ejemplos positivos y ninguno o pocos ejemplos negativos. En esta tesis, una hipótesis es una descripción de un concepto (como se verá a continuación) o un PTR en la forma de un conjunto de reglas (como se verá en la sección 4.6). Si el ejemplo satisface la descripción del concepto o PTR, se dice que el ejemplo es cubierto por la hipótesis. A continuación se describe cómo aprender el concepto *zona-frontal*.

1. **Generación de ejemplos positivos**. Se obtuvieron 922 ejemplos positivos en simulación colocando al robot en lugares donde existen obstáculos y registrando el estado. Las Figuras 4.6 (a), 4.6 (b) y 4.6(c) muestran al robot con obstáculos al frente en distintas posiciones. En vez de pares estado-acción se tienen pares estado,clase donde la clase es el nombre del concepto de zona que se quiere aprender:

$$
\begin{gathered}
zona(Estado_1, clase_1). \\
zona(Estado_2, clase_1). \\
zona(Estado_3, clase_1). \\
\vdots \\
zona(Estado_n, clase_1).
\end{gathered}
$$

   La Tabla 4.2 muestra tres ejemplos positivos para la *zona-frontal*. Cada ejemplo corresponde a las Figuras 4.6 (a), 4.6 (b) y 4.6 (c) respectivamente. El ejemplo (a)



(a) (b) (c)

Figura 4.6: Ejemplos positivos para el concepto *zona-frontal*

muestra un obstáculo a la derecha del robot a una distancia de 0.39 *metros*. Las dos marcas restantes registran otros obstáculos pero a una distancia cercana a tres metros por lo que no representan peligro, dichos obstáculos no se muestran en la figura. Los ejemplos (b) y (c) muestran un objeto cercano a la izquierda y al frente del robot.

(a) $zona\text{-}frontal([robot(0.71,7.00,316.42), posterior(\texttt{n}),$ $marca(2.78,\text{-}35.70,0.21,\texttt{i}), marca(2.97,\text{-}43.74,3.47,\texttt{p}),$ $marca(\mathbf{0.39},-45,75,3,13,\texttt{p})]$, `con-obstáculo`).
(b) $zona\text{-}frontal([robot(0.21,4.19,313.88), posterior(\texttt{n}),$ $marca(2.31,47.77,0.75,\texttt{i}), marca(1.43,79.94,0.28,\texttt{d}),$ $marca(3.06,46.76,3.50,\texttt{p}), marca(\mathbf{0.37},39.88,1.96,\texttt{p})]$, `con-obstáculo`).
(c) $zona\text{-}frontal([robot(3.29,4.71,260.68), posterior(\texttt{s}),$ $marca(2.71,\text{-}16.59,0.00,\texttt{e}), marca(\mathbf{0.35},\text{-}1.00,2.55,\texttt{p}),$ $marca(8.00,83.97,0.84,\texttt{p})]$, `con-obstáculo`).

Tabla 4.2: Ejemplos positivos: Concepto: *zona-frontal*

2. **Generación de ejemplos negativos**. Se obtuvieron 933 ejemplos para *zona-frontal* con la clase `sin-obstáculo` colocando al robot en zonas que no corresponden al concepto que se quiere aprender. Para este caso se coloca al robot en lugares libres de obstáculos o que se encuentran lejos del robot. En el ejemplo (a) el obstáculo más cercano se encuentra a una distancia de 0.87 *metros* y de acuerdo a quien guía al robot aún no lo considera peligroso. En el ejemplo (b) todos los obstáculos están a más de 2 *metros* de distancia por lo que no representan ningún riesgo (ver Fig 4.7 (b)). La Tabla 4.3 muestra dos ejemplos negativos.

3. **Conocimiento del dominio**. Es posible agregar conocimiento que pueda ser útil para inducir la hipótesis. El aprendizaje del concepto *zona-frontal* básicamente consiste en determinar los rango de distancia y orientación a los que una marca es considerada

(a) (b)

Figura 4.7: Ejemplos negativos para el concepto *zona-frontal*

(a) $zona\text{-}frontal([robot(0.55,6.82,313.02), posterior(\texttt{s}),$ $marca(2.93,\text{-}35.70,0.26,\texttt{i}), marca(3.14,\text{-}46.76,3.50,\texttt{p}),$ $marca(0.87,22.74,3.37,\texttt{p})], \texttt{sin-obstáculo}).$
(b) $zona\text{-}frontal([robot(\text{-}2.75,4.80,260.28), posterior(\texttt{n}),$ $marca(2.78,81.96,3.87,\texttt{d}), marca(2.76,9.55,0.00,\texttt{e}),$ $marca(4.65,82.96,0.58,\texttt{p})], \texttt{sin-obstáculo}).$

Tabla 4.3: Ejemplos negativos: Concepto *zona-frontal*

un obstáculo peligroso, por lo que se incluyeron las siguientes relaciones de igualdad para expresar los rangos:

$$menor\text{-}igual(X, Y) \leftarrow$$
$$X =< Y.$$

$$mayor(X, Y) \leftarrow$$
$$X > Y.$$

y también el predicado *obst-mas-cercano*($Estado$,$Ángulo$, $Distancia$) cuya entrada es el estado y cuya salida son el $Ángulo$ y la $Distancia$ hacia la marca más cercana.

El predicado objetivo consiste en una cabeza cuyos argumentos pueden ser: de entrada (+argumento), de salida (-argumento) o constantes (#argumento). En este caso, el argumento de entrada es el estado y la salida es una constante (#zona). El cuerpo está formado por los predicados que pueden aparecer en la hipótesis. En el predicado meta se consideran rangos de distancia y ángulos para la marca más cercana. No necesariamente el predicado aprendido incluirá todos los predicados dados por el usuario, como se verá en el resultado final. La notación que se muestra es como la usada en ALEPH.

$$
\begin{array}{l}
zona\text{-}frontal(+estado, \#zona). \\
\quad obst\text{-}mas\text{-}cercano(+estado,\text{-}ángulo,\text{-}distancia), \\
\quad menor\text{-}igual(+distancia, \#dvalor_1), \\
\quad menor\text{-}igual(+ángulo, \#angvalor_1), \\
\quad mayor(+ángulo, \#angvalor_2).
\end{array}
$$

4. **Aprendiendo las reglas con** ALEPH. Una vez obtenidos los ejemplos positivos, negativos, definido el posible conocimiento del dominio y el predicado meta, se dan como entrada al sistema de *ILP*, que realiza el siguiente proceso de aprendizaje:

   - De la traza, ALEPH selecciona un ejemplo como el siguiente:

$$
\begin{array}{l}
zona\text{-}frontal([robot(\text{-}2.83,5.24,260.28), posterior(\texttt{n}), \\
\qquad marca(2.3,11.56,0.0,\texttt{e}), marca(3.5,\text{-}61.84,0.0,\texttt{e}), \\
\qquad marca(2.82,\text{-}90.0,1.61,\texttt{p}), marca(2.15,\text{-}9.55,3.46,\texttt{p}), \\
\qquad marca(0.85,79.94,2.22,\texttt{p})],\ \texttt{sin-obstáculo}).
\end{array}
$$

   - ALEPH construye la cláusula más específica que es verdadera para el ejemplo seleccionado:

$$
\begin{array}{l}
zona\text{-}frontal(Estado,\texttt{sin-obstáculo}) \leftarrow \\
\quad obst\text{-}mas\text{-}cercano(Estado, A, D), \\
\quad menor\text{-}igual(D,0.85), \\
\quad menor\text{-}igual(A,79.94).
\end{array}
$$

   - ALEPH realiza un proceso de búsqueda de una cláusula más general que la cláusula más específica. Ésto se hace buscando sub-conjuntos de literales de la cláusula más específica que tengan la mejor evaluación. El usuario definió por experimentación que una cláusula será aceptada si cubre al menos 100 ejemplos positivos y como máximo 10 ejemplos negativos.

     La Figura 4.8 muestra un árbol de búsqueda generado para este ejemplo en el que el valor de clase es `sin-obstáculo` (933 ejemplos positivos y 922 ejemplos negativos). Los valores indican el número de ejemplos positivos y negativos cubiertos por la cláusula. Aunque todas las cláusulas cubren el mínimo número de ejemplos positivos establecido, todas exceden el máximo número de ejemplos negativos aceptados.

     La Figura 4.9 muestra un árbol de búsqueda para otro ejemplo con el valor de clase con-obstáculo (922 ejemplos positivos y 933 ejemplos negativos. A diferencia

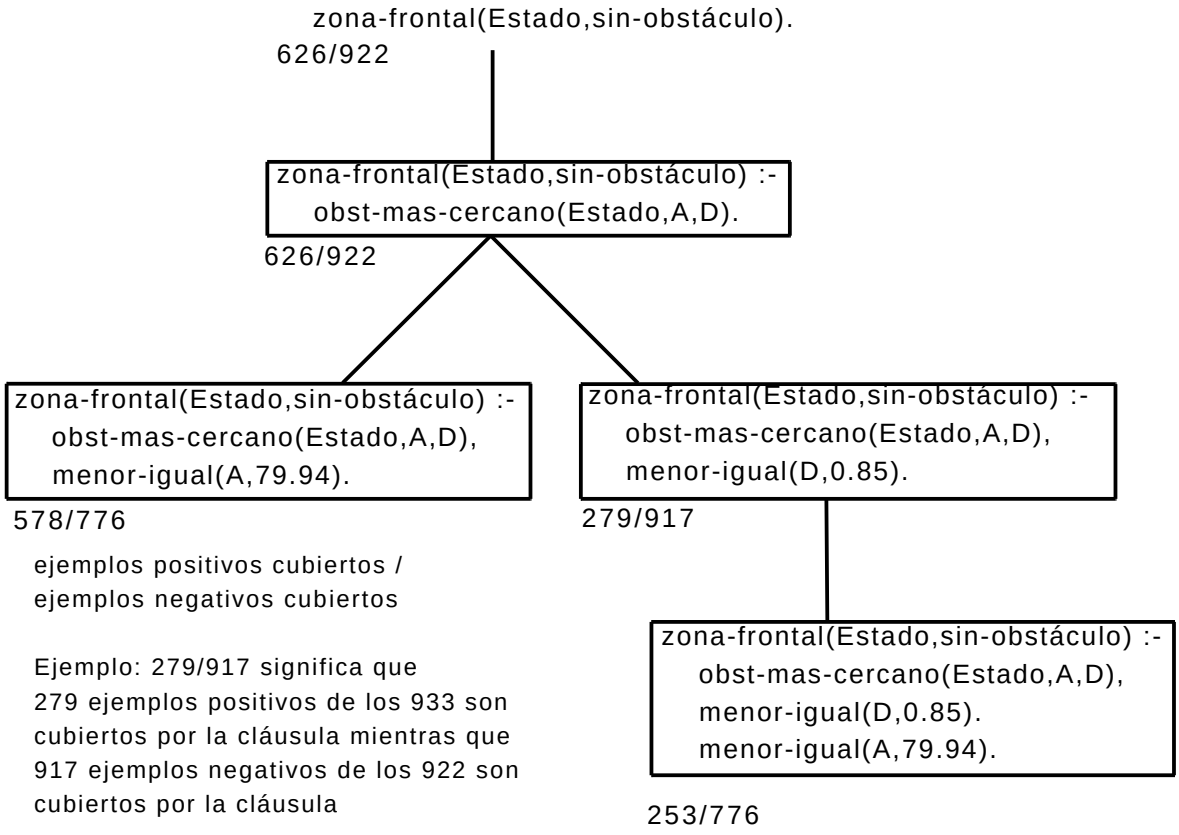


Figura 4.8: Búsqueda en ALEPH. Se muestra el árbol de búsqueda para un ejemplo, los números indican los ejemplos positivos y negativos cubiertos. Todas las cláusulas cubren más de los 100 ejemplos positivos establecidos como mínimo establecido para aceptar la cláusula pero exceden del máximo número de ejemplos negativos cubiertos permitido (máximo 10). Por ello, ninguna cláusula es aceptada.

del anterior, se construye una cláusula que cubre 891 ejemplos positivos y sólo 1 ejemplo negativo. Esta cláusula es agregada a la hipótesis y los ejemplos que cubre son eliminados del conjunto.

- El proceso se repite hasta que ya no hay ejemplos positivos que cubrir.

El concepto aprendido del proceso anterior es:

$$
\begin{aligned}
&\textit{zona-frontal}(\textit{Estado}, \texttt{con-obstáculo}) \leftarrow \\
&\quad \textit{obst-mas-cercano}(\textit{Estado}, A, D), \\
&\quad \textit{menor-igual}(D, 0.39). \\
&\textit{zona-frontal}(\textit{Estado}, \texttt{sin-obstáculo}).
\end{aligned}
$$

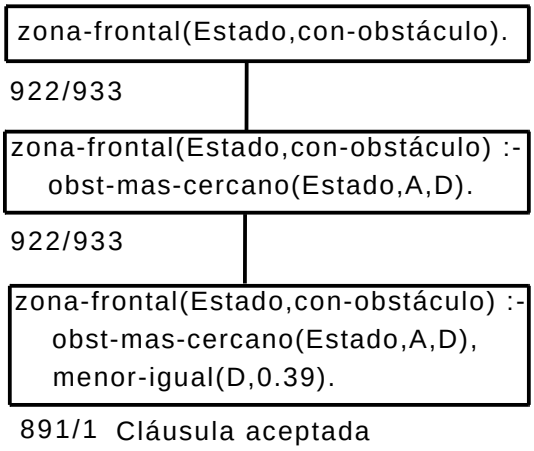


Figura 4.9: Búsqueda en ALEPH. ALEPH encuentra una cláusula que cubre 891 ejemplos positivos (al menos 100) y sólo un ejemplo negativo (máximo 10) por lo que cumple con los valores establecidos por el usuario. Es agregada a la hipótesis.

donde la entrada del predicado es la variable *Estado* y como salida puede tener los valores `con-obstáculo` ó `sin-obstáculo`. El predicado expresa que existe un obstáculo en la ***zona-frontal*** del robot si la marca más cercana se encuentra a 0.39 *metros* de distancia o menos; en caso contrario, la ***zona-frontal*** no tiene obstáculos. Como se observa, la relación de igualdad para la orientación de la marca no aparece en las cláusulas construidas lo cual significa que no importa la orientación de la marca, el único atributo relevante es su distancia relativa al robot.

Cuando el instructor guía al robot lo hace de forma muy particular, por ejemplo, una persona audaz evadirá obstáculos cuando esté muy cerca de éllos, a diferencia de una persona precavida quien los evadirá a una distancia mayor. Los estilos de operación se pueden capturar en las variables correspondientes a las distancias y orientaciones del láser y pueden aprenderse. La marca más cercana es una información exacta que no depende de apreciaciones, mientras que la distancia a la cual la persona considera necesario realizar una acción puede ser variable. En el ejemplo mostrado, para la persona que guió al robot, distancias mayores a 0.39 *metros* de los obstáculos son consideradas seguras para avanzar.

En [Likhachev y Arkin, 2000] se describe un trabajo relacionado al uso de zonas de "confort" en las que el robot "se sienta bien" para poder realizar determinadas acciones. La zona de "confort" es determinada por una función. Esta idea es similar al uso de zonas específicas para que el robot pueda ***deambular*** u ***orientarse*** que se aplica en esta tesis, aunque nosotros encontramos esas zonas usando aprendizaje computacional. El proceso de aprendizaje de conceptos se resume en los siguientes pasos:

1. Generar un conjunto de ejemplos positivos colocando al robot en la zona correspondiente al concepto que se quiere aprender

2. Generar un conjunto de ejemplos negativos colocando al robot en una zona distinta al concepto que se va a aprender

3. Agregar conocimiento del dominio si se dispone de él.

4. Aprender los conceptos usando ALEPH para un conjunto de ejemplos positivos, un conjunto de ejemplos negativos y posible conocimiento del dominio.

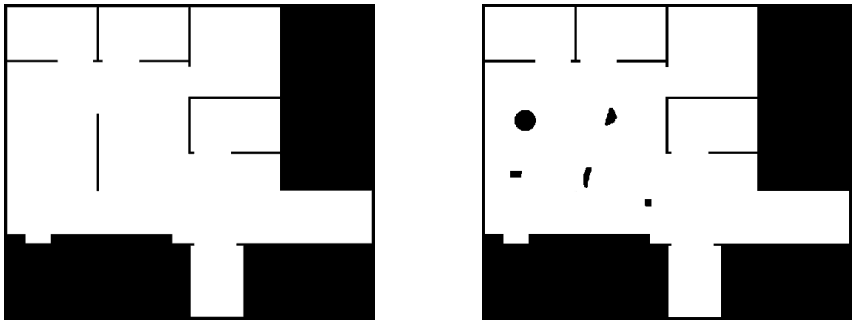
Para navegación se aprendieron diversos conceptos como: *zona-trasera*, que indica si existe obstáculo en la parte posterior del robot; la *zona-orienta*, que describe las condiciones para que el robot pueda orientarse; *distancia-pared*, que indica si la distancia hacia una pared es riesgosa o no; *config-paredes*, que describe si el robot está rodeado de paredes. Se definieron estos conceptos de forma intuitiva y una vez aprendidos los conceptos pueden usarse como conocimiento del dominio en el aprendizaje de los PTRs para las sub-tareas de navegación.

4.6 Aprendizaje de PTRs para sub-tareas de navegación

El aprendizaje de los PTRs para sub-tareas de navegación se realiza de forma similar al aprendizaje de conceptos. La diferencia es que en vez de ejemplos estáticos de zona se tienen trazas dinámicas, tomadas mientras se guía al robot a realizar la sub-tarea. El proceso de aprendizaje de PTRs para sub-tareas de navegación tiene por objetivo obtener un conjunto de predicados que den como salida una acción particular para realizar una sub-tarea específica.

Al igual que para el aprendizaje de conceptos se utiliza el esquema general de aprendizaje con *ILP*: se proporcionan ejemplos positivos, ejemplos negativos y posible conocimiento del dominio al sistema de *ILP* que inducirá las reglas de control (PTRs). A continuación se describe el proceso general de aprendizaje de PTRs:

1. **Generación de ejemplos positivos**. Para obtener las trazas, el usuario guía al robot usando el *joystick* en el simulador para realizar la sub-tarea a aprender. Se definieron dos ambientes de entrenamiento con elementos comunes en un ambiente de oficinas: habitaciones, paredes, pasillos, obstáculos fijos y obstáculos dinámicos que consisten en elementos que el usuario puede mover mientras guía al robot. Como obstáculo se



(a) (b)

Figura 4.10: Mapas de entrenamiento: (a) sin obstáculos, (b) con obstáculos. Como obstáculo se considera cualquier marca que se encuentre cerca del robot. La distancia considerada como "cerca" del obstáculo varía de acuerdo al usuario que entrene al robot.

considera cualquier marca que se encuentre cerca del robot. La cercanía se obtiene de las trazas del usuario al guiar al robot y evadir el obstáculo. Los mapas para entrenamiento fueron dibujados en un editor, sus medidas son de 14×12 m (ver Figura 6.2). La idea principal es que a partir del entrenamiento realizado en los mapas, los PTRs que se aprendan puedan ser usados en ambientes diferentes sin necesidad de repetir el entrenamiento.

A cada sub-tarea el usuario le asocia un conjunto de acciones, por ejemplo, para ***deambular***, el robot puede `avanzar`, `girar-a-la-izquierda` y `girar-a-la-derecha`. Cada acción puede ser utilizada por distintas sub-tareas.

El usuario guía al robot para obtener ejemplos de todo el conjunto de acciones. Las trazas con información de los sensores se transforman en predicados: a cada par estado-acción se le agrega el nombre del predicado (*i.e.*, nombre de la sub-tarea) que se va a aprender como se muestra a continuación:

$$sub - tarea(Estado, Acción).$$

La traza para una sub-tarea tiene la siguiente forma:

$$sub - tarea1(Estado_1, \texttt{acción}_3).$$
$$sub - tarea1(Estado_2, \texttt{acción}_1).$$
$$sub - tarea1(Estado_3, \texttt{acción}_1).$$
$$.$$

.

$$sub-tarea1(Estado_n,\texttt{acción}_2).$$

Las trazas de predicados obtenidas corresponden a los ejemplos positivos que el sistema de *ILP* necesita.

2. **Generación de ejemplos negativos**. Los ejemplos negativos pueden obtenerse: (i) generando trazas al ejecutar la sub-tarea de forma errónea, lo que es análogo a "mostrarle" al robot lo que no se debe hacer, o (ii) generarlos automáticamente a partir de los ejemplos positivos, intercambiando la acción. Para el caso del aprendizaje de PTRs para sub-tareas de navegación existen estados para los que puede aplicarse más de una acción, por lo que el intercambio se realiza con acciones claramente opuestas, como por ejemplo: `avanzar` por `retroceder` ó `girar-derecha` por `girar-izquierda`. La traza de ejemplos negativos a partir de la traza de ejemplos positivos tiene la siguiente forma:

$$sub-tarea1(Estado_1,\texttt{acción}_1).$$
$$sub-tarea1(Estado_2,\texttt{acción}_2).$$
$$sub-tarea1(Estado_3,\texttt{acción}_2).$$
$$.$$
$$.$$
$$sub-tarea1(Estado_n,\texttt{acción}_3).$$

3. **Conocimiento del dominio**. Al igual que en el aprendizaje de conceptos es posible incorporar conocimiento adicional tal como relaciones de igualdad (*e.g.*, *mayor-igual*, *menor-igual*), que se eligieron porque permiten indicar los rangos de valores y también los conceptos de zona aprendidos. En las secciones posteriores se describe con detalle.

4. **Aprendizaje de PTRs con** ALEPH. Los ejemplos positivos, los ejemplos negativos y el conocimiento del dominio son introducidos a ALEPH para el aprendizaje de los PTRs. El resultado es un conjunto de predicados con el formato que muestra el segundo renglón de la Tabla 4.4. Para cada estado, la salida es la acción a ejecutar. La idea es que para cada acción definida en el conjunto de acciones exista al menos un predicado. El algoritmo 4.1 resume el proceso. Adicionalmente, un PTR necesita una condición meta que indique cuándo la sub-tarea se ha realizado. Esta condición es agregada por

Algoritmo 4.1 Algoritmo para aprendizaje de PTRs básicos

Entrada: Conjunto de acciones A disponibles para la sub-tarea a aprender T, ejemplos positivos E^+ que son las trazas de alto-nivel obtenidas al guiar al robot realizando T, conocimiento del dominio Bkg

Variables locales: Conjunto de ejemplos negativos E^-

Salida: Programas Teleo Reactivos (PTRs) para realizar T

1: $E^- = \varnothing$ {Ejemplos negativos}
2: **for** all $e_i \in E^+$ **do** {De cada ejemplo positivo, generar un ejemplo negativo}
3: Cambiar la acción a en e_i por una opuesta {por ejemplo, izquierda por derecha, avanzar por retroceder}
4: Agregar e_i a E^-
5: **end for**
6: Sea Bkg la información adicional o aprendida disponible
7: Dados E^+, E^-, Bkg y una cláusula con predicado objetivo introducir a ALEPH para aprender PTRs

el usuario al conjunto de PTRs aprendidos y se muestra en el primer renglón de la Tabla 4.4.

Meta	$sub-tarea1(Estado, Acc) \leftarrow meta(Estado).$
PTR aprendido por ALEPH	$sub-tarea1(Estado, Acc) \leftarrow predicado1(Estado, ...),$ $predicado2(Estado, ...)$,...

Tabla 4.4: PTRs

Los PTRs aprendidos son probados en simulación y en un robot real. Para realizar las pruebas en simulación se construyen ambientes de prueba distintos a los de aprendizaje que incluyen elementos estáticos y dinámicos. Para realizar las pruebas con el robot se prepara un ambiente similar a un ambiente de oficina o de casa real, tomando en cuenta las limitaciones de los sensores (ver Capítulo 6).

Las siguientes secciones describen el aprendizaje de PTRs diversas sub-tareas de navegación: `deambular`, `orientarse`, `salir de una trampa` y `seguir` a un objeto móvil.

4.7 Aprendizaje del PTR para *deambular*

Cuando el robot se desplaza, la prioridad es preservar la seguridad tanto de las personas que están presentes en el ambiente como del robot mismo, así que iniciaremos con el aprendizaje del PTR al que llamaremos ***deambular***. En esta sub-tarea, el objetivo es que el robot

pueda alejarse de obstáculos a su paso y avance si no los hay. El robot carece de rumbo determinado y se moverá de acuerdo a lo que perciba. La meta del PTR es que el robot pueda realizar la acción `avanzar`.

El robot es guiado hacia una meta pasando por zonas con obstáculos y zonas libres hasta llegar a su destino. Se registran las lecturas de los sensores en trazas, generando así el conjunto de ejemplos positivos. Los ejemplos negativos se generan automáticamente a partir de los ejemplos positivos. Se agrega posible conocimiento del dominio ya sea proporcionado directamente por el usuario o aprendido. Los PTRs son inducidos usando ALEPH (Ver Figura 4.11).

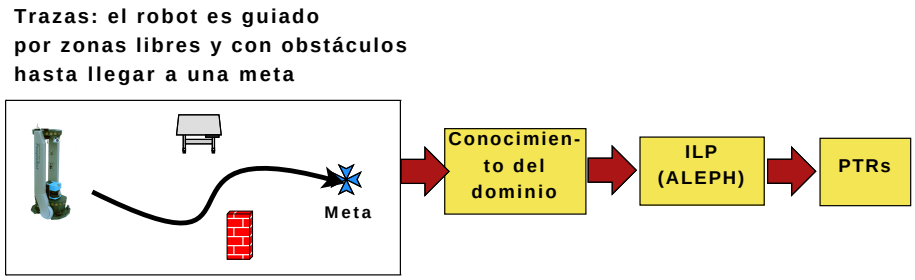


Figura 4.11: Aprendiendo a ***deambular***

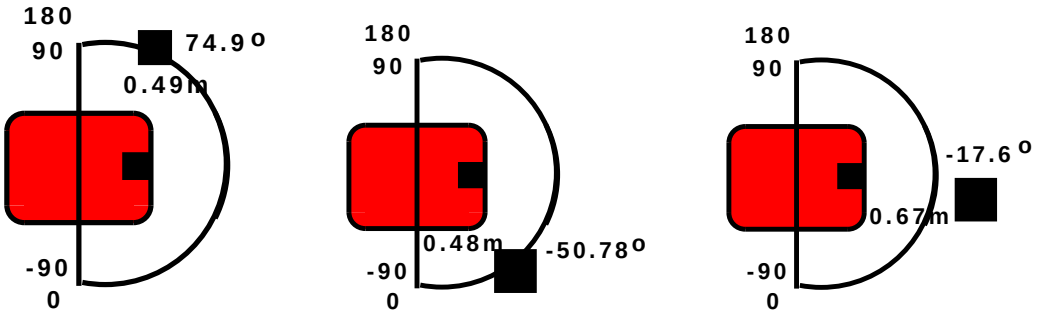
Ejemplos positivos. Durante el entrenamiento, el usuario evadió obstáculos fijos ubicados a diversas distancias y orientaciones del robot. Las marcas que percibía se consideraban obstáculos. Se generó una traza con 979 instancias distribuidas como sigue: `girar_derecha` (317), `girar_izquierda` (325) y `avanzar` (337). En la Tabla 4.5 vemos ejemplos positivos de evasión para las tres posibles acciones.

$deambular([robot(4.59,\text{-}1.86,335.39), posterior(\texttt{s}), meta(0,0), marca(0.49,74.92,2.83,\texttt{p}), ...]$,`girar_derecha`).
$deambular([robot(3.2,3.94,298.42), posterior(\texttt{n}), meta(0,0), marca(0.48,\text{-}50.78,3.64,\texttt{p}), ...]$,`girar_izquierda`).
$deambular([robot(3.03,3.89,332.32), posterior(\texttt{n}), meta(0,0), marca(0.67,\text{-}17.6,4.01,\texttt{p}), ...]$,`avanzar`).

Tabla 4.5: Ejemplos positivos: ***deambular***

Para el primer ejemplo, observamos que existe una marca a 0.49 *metros* de distancia ubicada 74.9 *grados*, lo que indica que está en la zona-izquierda (0-180) (ver Figura 4.12 (a)). Para evadir la marca, la acción fue girar a la derecha (`girar_derecha`). En el segundo ejemplo, la marca más cercana se encuentra a 0.48 *metros* del robot y a -50.78 *grados* (ver Figura 4.12 (b)). Para este caso, la acción fué girar a la izquierda (`girar_izquierda`). En

el tercer ejemplo, la marca más cercana se encuentra a 0.67 *metros* y la acción fue `avanzar` (ver Figura 4.12 (c)). Los ejemplos positivos muestran casos en los que la acción ejecutada es la correcta para el estado.



(a) (b) (c)

Figura 4.12: Ejemplos positivos: ***deambular***. En (a) el robot gira a la derecha, el obstáculo más cercano está a 0.49 m a una orientación de 74.9 $grados$ relativa al robot. En (b) el robot gira a la izquierda, el obstáculo más cercano se encuentra a -50.78 $grados$ del robot. En (c) el robot avanza pues el obstáculo más cercano aún no es riesgoso

Ejemplos negativos. Se generaron automáticamente tomando los ejemplos positivos e intercambiando los valores de clase. Para el aprendizaje de PTRs para sub-tareas de navegación, las acciones a intercambiar deben ser aquéllas que sean opuestas o muy diferentes entre si pues de esta forma, existen menos ambigedades para estados parecidos lo que mejora el resultado. En la Tabla 4.6 vemos los ejemplos negativos generados a partir de los ejemplos positivos anteriores. Otra forma de generar los ejemplos negativos es obtener trazas mientras se guía al robot de forma incorrecta, por ejemplo, al percibir una marca cerca, no alejarse y seguir avanzando. Sin embargo, al existir dentro de la misma traza distintas acciones, el intercambio de las acciones simplifica el trabajo.

$deambular([robot(4.59,\text{-}1.86,335.39), posterior(\texttt{s}), meta(0,0), marca(2.84,\text{-}5.53,0,\texttt{e}), ...]$,`girar_izquierda`).
$deambular([robot(3.2,3.94,298.42), posterior(\texttt{n}), meta(0,0), marca(3.47,31.68,0,\texttt{e}), ...]$,`avanzar`).
$deambular([robot(3.03,3.89,332.32), posterior(\texttt{n}), meta(0,0), marca(3.57,61.84,0,\texttt{e}), ...]$,`girar_derecha`).

Tabla 4.6: Ejemplos negativos: ***deambular***

Conocimiento del dominio y acciones. Tomando en cuenta que la información disponible de los sensores son distancias y orientaciones, lo más simple es expresar el conocimiento del

dominio en términos de rangos de ésos valores. En la Tabla 4.7 se muestra un ejemplo de posible conocimiento del dominio y las acciones para este PTR.

Conocimiento del dominio	Posibles valores de las variables
$zona\text{-}frontal(Estado, X)$ $zona\text{-}trasera(Estado, X)$	$X = \{$`sin-obstáculo`, `con-obstáculo`$\}$
$menor\text{-}igual(X, Y)$ $mayor\text{-}igual(X, Y)$	X,Y =$\{$*Distancia, Ángulo*$\}$
$obst\text{-}mas\text{-}cercano(Estado, Marca,$ $Distancia$,$Ángulo)$	

Tabla 4.7: Conocimiento del dominio ***deambular***

- ***zona_frontal*** y ***zona-trasera***: puede tener los valores: `con-obstáculo` o `sin-obstáculo`, indicando si frente al robot o atrás de él existe un obstáculo o no.

- *obst-mas-cercano*: obtiene la distancia y orientación del robot hacia la marca más cercana.

- *menor-igual* y *mayor-igual*: servirán para aprender los rangos de valores para distancia y ángulo hacia el obstáculo más cercano.

El problema de aprendizaje se traduce como: ¿cuál es el conjunto de reglas que combinando estos predicados genera la acción adecuada para evadir un obstáculo?

Una vez que han sido generados los ejemplos positivos y negativos y aprendidos, y los conceptos como conocimiento del dominio, se dan como entrada a ALEPH. El predicado tiene como entrada la variable *Estado* y como salida puede tomar los valores `avanzar`, `girar-izquierda` y `girar-derecha`. De la variable *Estado* se obtienen las variables $Marca$, $Distancia$ y $Angulo$ necesarias para el predicado $menor - igual$. El predicado ***deambular*** aprendido indica que el robot avanzará si en la ***zona-frontal*** no hay obstáculos; girará a la izquierda si en la ***zona-frontal*** hay obstáculo y está orientado a -1.51 *radianes*; en caso contrario, el robot girará a la derecha. A continuación se muestra el PTR expresado en Prolog:

```
deambular(Estado, avanzar) :-
    zona-frontal(Estado, sin-obstaculo).
deambular(Estado, girar-izquierda}) :-
    zona-frontal(Estado, con-obstaculo)
    obst-mas-cercano(Estado, Marca, Distancia, Angulo),
    menor-igual(Angulo, -1.51).
deambular(Estado, girar-derecha) :-
    zona-frontal(Estado, con-obstaculo).
```

Los PTRs se integran de forma manual, sin ninguna modificación al controlador para que el robot pueda deambular sin chocar con obstáculos.

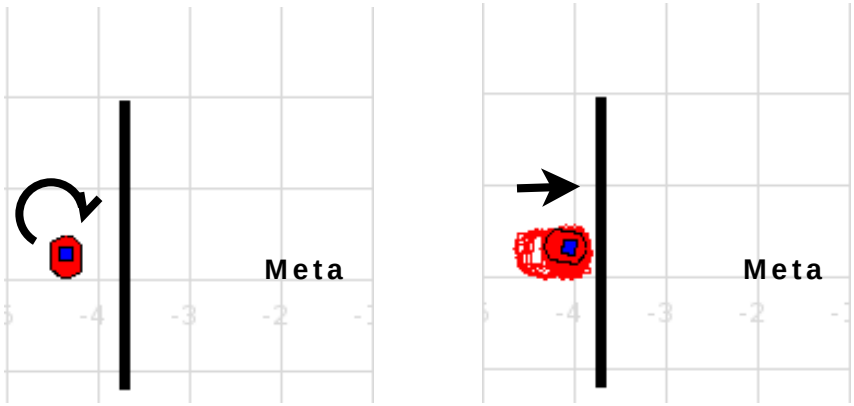
4.8 Aprendizaje del PTR para *orientarse*

Para que el robot pueda dirigirse a un punto determinado, necesita tener la capacidad de orientarse. En un ambiente libre de paredes o mobiliario, ésto sería suficiente para que el robot fuera capaz de llegar a la meta. Sin embargo, en un entorno real ésto no es tan simple. Si existe una pared larga entre el robot y la meta, con la simple habilidad de orientación el robot se mantendría orientando sin poder alcanzar el punto debido al obstáculo entre ambos. La Figura 4.13 muestra esta situación: en (a) el robot tiene la meta a su derecha, por lo que girará hacia ella, en (b) el robot trata de avanzar pero existe una pared que lo impide. El robot permanecerá girando para evadir la pared y orientándose hacia la meta indefinidamente.

Una estrategia simple para que el robot aprenda consiste en llevarlo a un área en la que no exista un obstáculo cercano entre él y la meta. Si existe un obstáculo entre ambos pero es lejano, el robot puede orientarse y avanzar sin problemas. En el momento en que el robot se encuentre cerca del obstáculo intermedio, buscará una zona libre de obstáculos entre el robot y la meta, para volver a orientarse.

Ejemplos positivos. Se obtuvieron ejemplos cubriendo los 4 cuadrantes del espacio con el fin de cubrir las posibles posiciones del robot relativas a la meta y que los predicados que se aprendan sean generales. Se produjo una traza con 3082 instancias: 1194 para la acción `girar-izquierda`, 993 para la acción `girar-derecha` y 895 para la acción `detenerse`.

Ejemplos negativos. La generación de ejemplos negativos se realizó intercambiando los valores de clase, de la misma forma que en los casos anteriores.



(a) (b)

Figura 4.13: (a) Robot con la meta a su derecha, (b) Se orienta y trata de ir hacia la meta

Conocimiento del dominio y acciones. El conocimiento del dominio para el PTR ***orientarse*** puede plantearse en términos de la distancia y orientación del robot hacia la meta y la posible existencia de obstáculos entre ambos (ver Tabla 4.8).

Conocimiento del dominio	Posibles valores de las variables
$zona\text{-}frontal(Estado, X)$ $zona\text{-}trasera(Estado, X)$	$X = \{$`sin-obstáculo`, `con-obstáculo`$\}$
$zona\text{-}orienta(Estado, X)$	$X = \{$`segura`, `no-segura`$\}$
$menor\text{-}igual(X, Y)$ $mayor\text{-}igual(X, Y)$	$X, Y = \{Distancia, Ángulo\}$
$obtener\text{-}angulos(Estado, Robot, Meta)$ $compara\text{-}angulos(Robot, Meta, X)$ $obtener\text{-}diferencias(Estado, Diferencia, Distancia)$	$Robot =$Ángulo del robot $Meta =$Ángulo de la meta $X = \{$`iguales`, `diferentes`$\}$

Tabla 4.8: Conocimiento del dominio para ***orientarse***

Los predicados del conocimiento del dominio para orientarse son los siguientes:

- ***zona_frontal***: puede tener los valores: `con-obstáculo` o `sin-obstáculo`, indicando si frente al robot existe un obstáculo o no.

- ***zona-orienta***: sus posibles valores son: `segura` y `no-segura` dependiendo si existe obstáculo cercano entre el robot y la meta.

- *obtener-ángulos*: extrae del estado las orientaciones del robot y la meta. A partir de la posición del robot y la posición de la meta, el ángulo de la meta ($angm$) se calcula mediante la función trigonométrica $acos(Valor)$. A continuación se muestra el cálculo:

$$CA = X_{meta} - X_{robot}$$
$$CO = Y_{meta} - Y_{robot}$$
$$dm = \sqrt{CA^2 + CO^2}$$

$$Valor = \frac{X_{meta} - X_{robot}}{dm} = \frac{CA}{dm}$$

$$angm = acos(Valor)$$

Ejemplo. La Figura 4.14 muestra un robot cuya posición es (3,2) y la meta se encuentra en la posición (5,5). Se calcula el ángulo de la meta, obteniéndose 56.63 *grados*. Para orientarse, el robot tiene que igualar ése ángulo.

$$CA = 2$$
$$CO = 3$$
$$dm = \sqrt{4 + 9} = 3{,}61$$

$$Valor = \frac{CA}{dm} = \frac{2}{3{,}61} = 0{,}55$$

$$angm = acos(0{,}55) = 56{,}63$$

- *compara-ángulos*: sus posibles valores son iguales o diferentes expresando si la orientación del robot y de la meta es la misma o no.

- *obtener-diferencias*: extrae del estado las distancias hacia las paredes y obtiene las diferencias entre distancias.

La meta del PTR es que el robot esté orientado hacia el punto destino y su acción meta es *detenerse*.

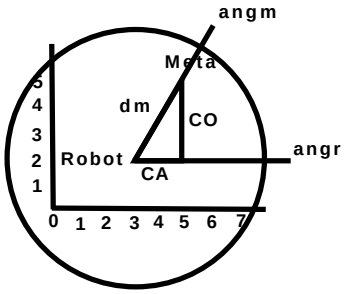


Figura 4.14: A partir de la posición del robot y la posición de la meta, el ángulo de la meta (angm) se calcula mediante la función trigonométrica $acos(Valor)$.

Aprendizaje de conceptos como conocimiento del dominio. Dependiendo del criterio del usuario que entrene al robot, los rangos de distancia que definen una zona de orientación pueden variar. Estos criterios pueden proporcionarse o aprenderse. El concepto aprendido es ***zona-orienta*** en el cual la entrada es la variable *Estado* y las posibles salidas son `segura` y `no-segura`. En el predicado correspondiente a conocimiento del dominio $obt-val$, la entrada es también la variable *Estado* y sus salidas son las variables *LongObst*, $Diferencia$ y $Distancia$. A continuación se muestran los predicados aprendidos:

$$
\begin{aligned}
&zona{-}orienta(Estado, no{-}segura) :- \\
&\quad obt{-}val(Estado, Diferencia, Distancia), \\
&\quad Atrib > 1, \\
&\quad Diferencia < 50, \\
&\quad Distancia <= 1. \\
&zona{-}orienta(Estado, no{-}segura) :- \\
&\quad zona{-}trasera(Estado, hay{-}obstaculo). \\
&zona{-}orienta(Estado, segura\}) :- \\
&\quad obt{-}val(Estado, LongObst, Diferencia, Distancia), \\
&\quad LongObst > 1, \\
&\quad Diferencia > 50, \\
&\quad zona{-}frontal(Estado, sin{-}obstaculo).
\end{aligned}
$$

donde la salida puede tomar los siguientes valores:

- `no-segura`: tiene dos predicados (1) si la diferencia entre la orientación del robot y la meta es menor a 50 *grados* (lo que indica que el robot está cercanamente orientado

hacia élla), el obstáculo mide más de 1 m, y la distancia entre el obstáculo y el robot es menor a 1 m (cercana) y (2) si hay un obstáculo en la parte trasera del robot pues si el obstáculo fuera móvil el robot podría chocar.

- `segura`: el robot está en una zona segura para ***orientarse*** hacia la meta si el obstáculo entre éllos mide más de 1 m, el robot no está orientado hacia la meta y el predicado *zona-frontal* tiene el valor `sin-obstáculos`. Ésto significa que aunque existiera un obstáculo intermedio, no estaría cercano por lo que el robot podría avanzar. Si no hay obstáculos, el robot puede orientarse hacia la meta.

Los ejemplos positivos y negativos, así como el conocimiento del dominio se proporcionaron a ALEPH y se aprendieron los predicados para orientación. La entrada es la variable *Estado* y los posibles valores de salida son `girar-izquierda`, `detenerse` y `girar-derecha`. El predicado $obten-angulos$ utiliza también la variable de entrada *Estado* para obtener las variables de salida *Robot* y *Meta*. A continuación se muestran los predicados para ***orientarse***:

```
orientarse(Estado, girar-izquierda) :-
    zona-orienta(Estado, segura),
    obten-angulos(Estado, Robot, Meta),
    compara-angulos(Robot, Meta, diferentes),
    meta-izquierda(Robot, Meta).
orientarse(Estado, detenerse) :-
    zona-orienta(Estado, segura),
    obten-angulos(Estado, Robot, Meta),
    compara-angulos(Robot, Meta, iguales).
orientarse(Estado, girar-derecha) :-
    zona-orienta(Estado, segura).
```

con los siguientes posibles valores:

- `girar-izquierda`: el predicado toma este valor cuando *zona-orienta* es segura, la orientación de la meta y el robot son diferentes, y la meta se localiza a la izquierda del robot.

- `detenerse`: el predicado toma este valor cuando ángulo del robot y el ángulo de la meta son el mismo.

- `girar-derecha`: el predicado toma este valor cuando la meta se encuentra a la derecha del robot y *`zona-orienta`* toma el valor `segura`.

4.9 Aprendizaje del PTR para *salir de una trampa*

En un ambiente doméstico, es muy común encontrar espacios estrechos en los cuales el robot puede entrar pero de los cuales es difícil salir. La Figura 4.15 muestra un ejemplo en el cual el robot se encuentra orientado hacia la meta y percibe espacio libre frente a él. Lo que el robot hace en tal caso es avanzar pues percibe espacio suficiente. Sin embargo, el robot no es capaz de identificar que las paredes están en una configuración que no le permitirá salir. Cuando ya no puede avanzar, aún tiene espacio para girar a la derecha pero ya no hay espacio suficiente para que dé la vuelta y salga por lo que el robot queda atrapado.

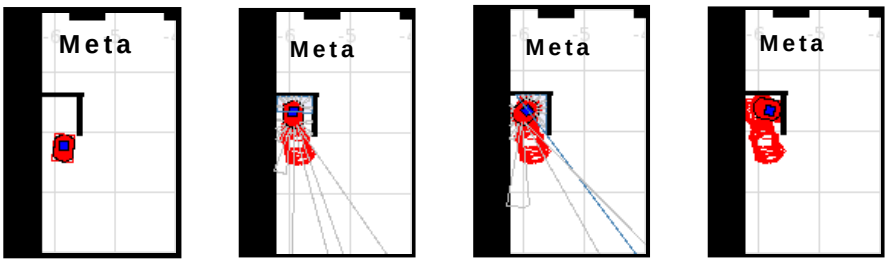
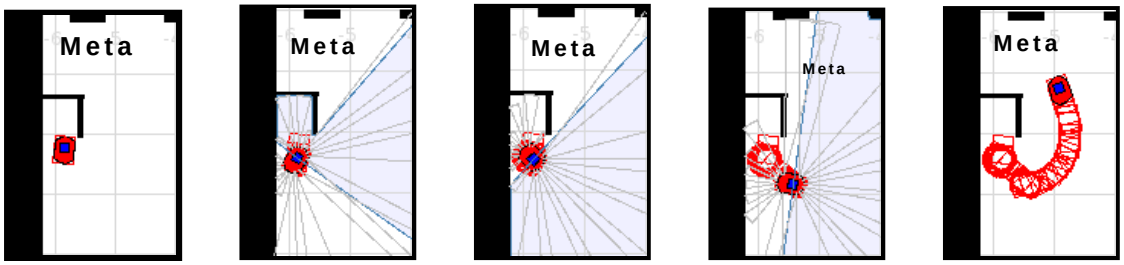


Figura 4.15: Robot atrapado: (a) posición inicial, (b) hay espacio al frente y puede avanzar, (c) no puede seguir porque encuentra pared y gira a la derecha, (d) el robot queda atrapado

Lo esperado es que el robot perciba que se encuentra en una trampa y sea capaz de salir de ella. En este caso, estamos usando un enfoque reactivo por lo que para reconocer una trampa, el robot tiene que entrar y reconocerla para poder salir. Considerando ésto, el PTR debe tener dos funciones: (i) identificar la trampa y (ii) salir de élla. La Figura 4.16 muestra al robot utilizando el PTR ***salir de trampa***. El robot se orienta hacia la meta pero reconoce que está en una trampa, sale de ella y vuelve a ***orientarse*** hacia la meta. La estrategia no garantiza que el robot podrá salir en todos los casos, por ejemplo, en casos donde el ambiente cambia, presentándose un obstáculo que forma una trampa después de que el robot entró a un lugar, el robot difícilmente se liberará. También le resultará difícil o imposible salir si la distancia aprendida como `riesgosa` es muy pequeña ($e.g.$, 0.2 m) pues no detectará la trampa hasta después de haberse introducido a ella. Sin embargo, como se verá en los resultados experimentales, agregar este PTR mejora la precisión en la tarea de

navegación al reducir los casos en que el robot se queda atrapado y logra llegar a la meta.



(a) (b) (c) (d) (e)

Figura 4.16: Robot identificando una trampa y saliendo de élla. (a) Posición inicial, (b) Se orienta y trata de ir hacia la meta

Ejemplos positivos. Se generó una traza del robot al guiarlo para que saliera de una trampa ubicándolo en distintas posiciones iniciales y con distinta profundidad dentro de la trampa. La traza incluye 237 instancias para la acción `retroceder`, 235 para la acción `girar-derecha`, 235 `girar-izquierda` y 245 para la acción `detenerse`.

Ejemplos negativos. De manera similar a los PTRs anteriores, los ejemplos negativos se generaron intercambiando los valores de clase.

Conocimiento del dominio y acciones. Una vez definida la sub-tarea que se quiere aprender, el usuario proporciona el posible conocimiento del dominio y las acciones para ***salir de trampa*** (ver Tabla 4.9).

Conocimiento del dominio	Posibles valores de las variables
$config\text{-}paredes(Pared\text{-}der, Pared\text{-}izq, Pared\text{-}frente, X)$	$X = \{$`libre-trampa`,`en-trampa`,`dejando-trampa`$\}$
$lado\text{-}pared(Pared\text{-}der, Pared\text{-}izq, X)$	$X = \{$`libre-derecha`, `libre-izquierda`$\}$
$distancia\text{-}pared(Pared, X)$	$X = \{$`riesgosa`, `segura`$\}$
$zona\text{-}frontal(Estado, X)$	$X = \{$`sin-obstáculo`, `con-obstáculo`$\}$
$menor\text{-}igual(X, Y)$ $mayor\text{-}igual(X, Y)$	$X, Y = \{Distancia, Ángulo\}$

Tabla 4.9: Conocimiento del dominio para ***salir de trampa***

donde

- *config-paredes*: recibe las distancias del robot hacia las paredes izquierda, derecha y frontal. Sus posibles valores son `en-trampa`, `libre-trampa` o `dejando-trampa` para determinar si el robot está o no en una trampa o está saliendo de élla.

- *lado-pared*: recibe las distancias hacia las paredes del lado derecho e izquierdo del robot. Sus posibles valores son `libre-derecha` y `libre-izquierda` para saber hacia dónde puede girar el robot para salir.

Dependiendo de la guía del usuario, las distancias entre las paredes que pueden considerarse una trampa pueden variar. Un espacio será una trampa para un usuario y no lo será para otro. Estos criterios de distancia se pueden aprender. El predicado *config-paredes* captura esta información, pero requiere que se aprenda previamente el concepto ***distancia-pared***. La entrada del predicado *config-paredes* son las variables $Pared{-}der$, $Pared{-}izq$ y $Pared{-}frente$ que son las distancias hacia las paredes. La salida puede tomar los valores que se muestran a continuación:

- *en-trampa*: el robot se encuentra en una trampa cuando el predicado ***distancia-pared*** toma el valor `riesgosa` para las paredes que se encuentran a su derecha, a su izquierda y frente a él. En este caso, por el aprendizaje del concepto ***distancia-pared*** a partir de trazas de un usuario, la distancia se considera `riesgosa` cuando es menor-igual a 0.7 m.

- *libre-trampa*: el robot no se encuentra en una trampa cuando el predicado ***distancia-pared*** toma el valor `segura`, en este caso, cuando la distancia es *mayor-igual* a 0.7 m. Los conceptos ***config-paredes*** y ***distancia-pared*** aprendidos son los siguientes:

```
config-paredes(Pared-der, Pared-izq, Pared-frente, entrando-trampa) :-
    distancia-pared(Pared-der, riesgosa),
    distancia-pared(Pared-izq, riesgosa),
    distancia-pared(Pared-frente, riesgosa).
config-paredes(Pared-der, Pared-izq, Pared-frente, libre-trampa) :-
    distancia-pared(Pared-der, segura),
    distancia-pared(Pared-izq, segura)).
    config-paredes(_, _, _, dejando-trampa).
```

```
distancia-pared(Pared, riesgosa) :-
    distancia(Pared, Dist),
    menor-igual(Dist, 0.7).
distancia-pared(Pared, segura) :-
    distancia(Pared, Dist),
    mayor-igual(Dist, 0.7).
```

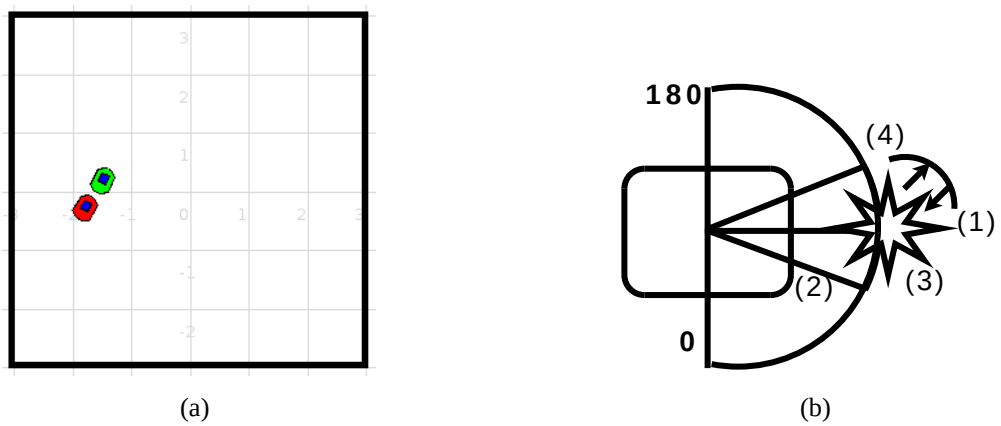
donde la entrada al predicado $distancia - pared$ es las variable $Pared$ que representa la distancia del robot hacia una pared. Los posibles valores de salida son `riesgosa` y `segura`, dependiendo de la cercanía de la pared hacia el robot.

Las trazas y el conocimiento del dominio fueron dados a ALEPH y se aprendieron los siguientes PTRs para ***salir de trampa***:

```
salir-de-trampa(Estado, retroceder) :-
    config-paredes(Estado, Pared-der, Pared-izq, Pared-frente,
                                                  entrando-trampa).
salir-de-trampa(Estado, detenerse) :-
    config-paredes(Pared-der, Pared-izq, Pared-frente, libre-trampa),
    zona-frontal(Estado, sin-obstaculo).
salir-de-trampa(Estado, girar-derecha) :-
    lado-pared(Pared-der, Pared-izq, libre-derecha),
salir-de-trampa(Estado, girar-izquierda) :-
    lado-pared(Pared-der, Pared-izq, libre-izquierda).
```

donde la entrada es la variable *Estado* y la variable de salida puede tomar los siguientes valores :

- `retroceder`: el robot se encuentra en una trampa.
- `detenerse`: el robot no se encuentra en una trampa.
- `girar-derecha`: el robot percibe que a la derecha tiene espacio para girar y salir.
- `girar-izquierda`: el robot percibe que a la izquierda tiene espacio para girar.



(a) (b)

Figura 4.17: (a) Ambiente de entrenamiento: el robot-aprendiz es guiado usando un *joystick*. El robot que es seguido usa el PTR aprendido para deambular. (b) Para aprender a seguir, se utiliza el rango del láser (1) para aprender la zona de seguimiento, la distancia (2) del robot al objeto (3), e identificar el movimiento del objeto (4) para seguirlo.

4.10 Aprendizaje del PTR para *seguir* a un objeto móvil

Esta sub-tarea consiste en aprender a seguir a un objeto móvil utilizando el sensor láser. Se definió una zona de seguimiento frente al robot en la que éste identifique al objeto más cercano, representado por la marca más cercana a la cual seguirá. El objetivo es que una vez que el robot haya identificado un objeto en la zona de seguimiento, se mantenga a una distancia cercana de él. Para lograrlo, el robot tiene que desplazarse en la misma dirección que el objeto, avanzando, girando a la izquierda ó a la derecha ó deteniéndose si el objeto se para. Aunque este enfoque es muy básico, una de sus principales ventajas fue que el PTR que se aprendió pudo ser utilizado reemplazando el sensor láser con una cámara y mediante un proceso de reconocimiento de marcas visuales envía la marca del objeto a seguir (ver , Figura 4.17).

Ejemplos positivos. El escenario de entrenamiento consiste en un área libre en la cual el robot es guiado para seguir a otro robot que es controlado por el PTR aprendido ***deambular*** (ver Figura 4.17(a)). Se obtuvieron 619 ejemplos, de los cuales 117 corresponden a la acción `girar-derecha`, 147 son de la acción `avanzar`, 117 ejemplos para la acción `girar-izquierda`, 118 ejemplos para la acción `detenerse` y 120 ejemplos para `retroceder`. Al robot se le guió para retroceder cuando el objeto al que sigue se aproxima a él. Avanzará para seguir al objeto que se encuentre en la zona de seguimiento, girará para seguirlo y

Conocimiento del dominio	Posibles valores de las variables
$zona\text{-}seguir(ÁnguloObjeto, X)$	$X = \{$`dentro`,`fuera`$\}$
$rango\text{-}distancia(Distancia, X)$	$X = \{$`cerca`, `lejos`$\}$
$menor\text{-}igual(X, Y)$ $mayor\text{-}igual(X, Y)$	$X, Y = \{Distancia, Ángulo\}$
$obten\text{-}val(Estado$,Á$nguloObjeto, Distancia)$	
Acciones para la sub-tarea	`detenerse`, ,`avanzar`, `girar-izquierda`, `girar-derecha`, `retroceder`

Tabla 4.10: Conocimiento del dominio y acciones para ***seguir***

se detendrá cuando esté cerca.

Ejemplos negativos. Los ejemplos negativos se generaron intercambiando las clases.

Conocimiento del dominio y acciones. En la Tabla 4.10 se muestran el conocimiento del dominio y las acciones para el PTR de seguimiento. Se definieron los siguientes predicados:

- *obten-val*: obtiene la distancia y el ángulo relativos al robot de la marca más cercana.
- ***zona-seguir***: sus posibles valores son `dentro` y `fuera` dependiendo si el objeto más cercano se encuentra dentro de la zona de seguimiento o no.
- *rango-distancia*: sus posibles valores son: `cerca`, `lejos` o `riesgosa` dependiendo de la distancia del robot a la marca a seguir.
- *menor-que* y *mayor-que* se usan para aprender los rangos del láser que definen la zona de seguimiento.

Conceptos como conocimiento del dominio. Para el PTR de seguimiento, pueden aprenderse los rangos de la zona en la cual debe estar el objeto para que el robot lo siga. El rango de valores de la zona de seguimiento variará dependiendo del criterio de cada usuario para determinar dónde debe estar el objeto para decidir seguirlo. Esta zona está definida por el rango de lecturas del sensor láser. A continuación se muestra el predicado aprendido:

$$
\begin{aligned}
&zona{-}seguir(AnguloObjeto, dentro) :- \\
&\quad AnguloObjeto < 137.77, \\
&\quad AnguloObjeto > 49.27.
\end{aligned}
$$

donde la entrada es la variable *AnguloObjeto* y la variable de salida puede tomar los valores `dentro` o `fuera` dependiendo de que el objeto esté o no en la zona de seguimiento. El rango de la zona de seguimiento va de 49.27 *grados* a 137.77 *grados* de acuerdo al usuario que guió al robot.

A continuación se muestran los PTRs para ***seguir*** a un objeto móvil. La variable de entrada es *Estado* de la cual otros predicados como $obten-val$ obtienen variables de salida como *Ang* y *Dist*. La salida puede tomar los valores `retroceder`, `detenerse`, `girar-derecha`, `girar-izquierda` y `avanzar`. El primer predicado indica que la acción a realizar será retroceder si la distancia hacia la marca es riesgosa. Éso significa que el objeto se está aproximando hacia el robot y tiene que retroceder para mantener una distancia mínima. En el segundo predicado, el robot se detendrá si la distancia hacia el objeto es cercana. El tercer predicado indica que si el objeto está en zona de seguimiento pero a una orientación menor a 70.39 *grados*, el robot debe girar hacia la derecha. Si el robot se encuentra dentro de la zona pero en una orientación mayor a 105.59 *grados* entonces el robot debe girar hacia la izquierda. El último predicado expresa que el robot va a avanzar si el objeto está dentro de la zona.

```
seguir(Estado, retroceder) :-
    obten-val(Estado, Ang, Dist),
    rango-distancia(Dist, riesgosa).
seguir(_, Dist, Ang, detenerse) :-
    zona-seguir(Ang, dentro),
    rango-distancia(Dist, cerca).
seguir(Estado, girar-derecha) :-
    obten-val(Estado, Ang, Dist),
    zona-seguir(Ang, dentro),
    menor-que(Ang, 70.39).
seguir(Estado, girar-izquierda) :-
    zona-seguir(Ang, dentro),
    mayor-que(Ang, 105.59).
seguir(Estado, avanzar) :-
    zona-seguir(Ang, dentro).
```

El PTR aprendido controla al robot con una velocidad constante por lo que si el objeto se desplazara un poco más rápido el robot lo perdería con facilidad. Se codificaron manualmente

variaciones en la velocidad, de manera que a mayor distancia del objeto a seguir, la velocidad del robot es más alta. Éstas variaciones de velocidad pueden también aprenderse.

4.11 Discusión

La definición del conocimiento del dominio es uno de los aspectos clave al usar programación lógica inductiva. El conocimiento del dominio consiste en definiciones de predicados generales que pueden usarse en las hipótesis inducidas.

Una ventaja es que el conocimiento del dominio permite al usuario intervenir en el proceso de aprendizaje y sus resultados. Proporciona flexibilidad al usuario si tiene conocimiento parcial, intuiciones o expectativas sobre la hipótesis a ser inducida. Si no se tiene, el espacio de búsqueda será mayor, puede necesitar más ejemplos y tiempo y en el peor de los casos el aprendizaje puede no tener éxito [Džeroski et al., 2001].

El número y la calidad de los ejemplos son otro de los aspectos críticos en este tipo de aprendizaje. Pueden darse muchos ejemplos bajo situaciones parecidas lo que originará baja precisión al presentarse al robot nuevos estados. Por el contrario, si se dan pocos ejemplos aunque bajo situaciones diversas, la generalización puede ser deficiente. Con un balance adecuado entre número de ejemplos y diversidad de situaciones se logra aprender reglas generales que permiten al robot realizar la sub-tarea. Sin embargo, encontrar este balance no siempre es sencillo.

La calidad de los ejemplos es importante pues las reglas inducidas reproducirán la forma en que el usuario guió al robot. No es necesario que los ejemplos dados sean perfectos pues es común que los algoritmos de *ILP* manejen ruido al aceptar porcentajes de ejemplos negativos cubiertos por las reglas. Al guiar al robot con el *joystick* en el simulador es común que el usuario se equivoque o que el simulador genere algún error en la lectura. Sin embargo, el usuario puede determinar el número de ejemplos positivos y negativos que las reglas pueden cubrir.

4.12 Resumen

En este capítulo se describió el proceso de aprendizaje de PTRs básicos para navegación de un robot móvil. El proceso de aprendizaje consta de diversas etapas: transformación de información de bajo nivel a una representación relacional, definición de los PTRs a aprender,

aprendizaje del conocimiento del dominio, generación de ejemplos negativos, y por último el aprendizaje con el sistema de *ILP* ALEPH que da por resultado un conjunto de PTRs básicos.

La representación del ambiente se obtiene por un proceso automático de transformación de niveles de abstracción. Cuando el usuario guía al robot a realizar la tarea o sub-tareas, las trazas con información de bajo nivel de los sensores se transforman por un proceso de identificación de marcas naturales a una representación de alto nivel.

Los PTRs de navegación definidos son: ***deambular***, ***orientarse***, ***salir de una trampa*** y ***seguir*** a un objeto móvil. Para cada una de éllas se especificó un conjunto de acciones disponibles y conocimiento del dominio a aprender. Se aprendieron conceptos como conocimiento del dominio para cada sub-tarea. Estos conceptos corresponden generalmente a zonas del ambiente, por ejemplo zona-frontal, que indica si frente al robot hay o no un obstáculo. Se generaron ejemplos negativos de forma automática a partir de los ejemplos positivos. Para cada ejemplo positivo, la clase era cambiada por otra distinta dentro de su conjunto de posibles acciones.

Los ejemplos positivos, los ejemplos negativos y el conocimiento del dominio aprendido se dió como entrada a ALEPH, obteniendo por resultado un conjunto de PTRs básicos para cada sub-tarea definida. La Tabla 4.11 resume el proceso de aprendizaje. La primer columna muestra el nombre del PTR aprendido, la columna E+ muestra el número de ejemplos positivos. En la columna Conoc-Dom se muestra el conocimiento del dominio que es aprendido(*) y el que es dado por el usuario. Existe conocimiento del dominio que es usado por varios PTRs

Una vez aprendidos los PTRs básicos, se pueden generar secuencias de éllos al guiar al robot a realizar tareas que usen esos PTRs. A partir de ésas secuencias se aprenden los PTRs compuestos usando el algoritmo para aprendizaje de gramáticas que se describe en el siguiente capítulo.

PTR	E+	Conoc-Dom	Acciones
deambular	979	*zona-frontal** *zona-trasera** $menor-igual$,$mayor-igual$ *obst-mas-cercano*	avanzar girar-derecha girar-izquierda
orientarse	2982	$obtener-angulos$ *zona-orienta** *zona-frontal** *zona-trasera** $menor-igual$, $mayor-igual$ $obst-mas-cercano$	detenerse,girar-izquierda girar-derecha avanzar girar-derecha
salir-de-trampa	952	$config-paredes$* $lado-pared$* *zona-frontal** $obst-mas-cercano$	detenerse,girar-izquierda girar-derecha avanzar retroceder
seguir	619	$obten-vals$ *zona-seguir** $rango-distancia$*	detenerse, avanzar girar-derecha,girar-izquierda retroceder

Tabla 4.11: PTRs aprendidos, donde (*) significa que el concepto fue aprendido

5

FOSeq: Aprendizaje Relacional de Gramáticas

5.1 Introducción

Muchas tareas pueden ser descritas por una secuencia de acciones para lograr un objetivo o meta. Ésas secuencias pueden en ciertos casos ser representadas en una estructura jerárquica que puede usarse para realizar o reconocer la tarea. Generalmente un robot móvil ejecuta una secuencia de acciones cuando realiza una tarea. En este capítulo se presenta FOSeq, un algoritmo que aprende gramáticas a partir de secuencias de acciones. Las secuencias son dadas como trazas de bajo nivel de datos de sensores que son transformados en una representación de alto nivel. FOSeq induce una gramática para cada secuencia y realiza un proceso de generalización sobre la mejor gramática para cubrir la mayoría de las secuencias. FOSeq es capaz de aprender gramáticas a partir de secuencias de símbolos o de secuencias de predicados de primer orden. Las gramáticas inducidas por FOSeq pueden ser usadas para ejecutar una tarea particular o para clasificar nuevas secuencias. El enfoque se usa en dos dominios: (i) navegación y (ii) reconocimiento de ademanes.

5.2 FOSeq: Algoritmo general

El objetivo de FOSeq (*First Order Sequence learning*) es aprender PTRs que sean capaces de generar secuencias particulares de acciones que satisfagan una meta. En este caso, los PTRs pueden expresarse en términos de otros PTRs. Al igual que para el aprendizaje de PTRs básicos, se necesitan ejemplos de la tarea. El usuario guía al robot para realizar una tarea, por

ejemplo, ir a un lugar específico. La información de bajo nivel de los sensores es transformada en trazas de alto nivel. Utilizando el mismo proceso de identificación de marcas descrito en la sección 4.3 la información de bajo nivel proveniente de los sensores es transformada a una representación de marcas como muestra la Tabla 5.1.

Traza de marcas	([$robot$(-4.93,-0.96,357.56), $atrás$(no), $meta$(-1.00,3.00), $marca$(4.78,-33.69,1.02, e), , ..., $marca$(6.32,88.99,0.37,p)]). ([$robot$(-4.93,-0.96,355.76), $atrás$(no), $meta$(-1.00,3.00), $marca$(4.85,-34.69,1.03,i), ..., $marca$(6.33,86.98,0.35,p)]). ...
Traza de PTRs	$orientarse$($Estado_1$,girar-izquierda), $deambular$($Estado_2$,avanzar), $deambular$($Estado_3$,avanzar),$deambular$($Estad_4$,girar-izquierda)...

Tabla 5.1: Ejemplo de trazas de entrada. El usuario guía al robot para realizar la tarea y obtiene trazas con la posición del robot y las marcas que percibe. Utilizando el mismo proceso de identificación de marcas descrito en la sección 4.3 la información de bajo nivel proveniente de los sensores es transformada a una representación de marcas. Las trazas están formadas por estados que se dan como entrada a al conjunto de PTRs aprendidos codificados como un programa en Prolog. Para cada estado, el programa imprime el PTR y la acción que aplica, generando una traza de PTRs.

Las trazas están formadas por estados que se dan como entrada a al conjunto de PTRs aprendidos codificados como un programa en Prolog. Para cada estado, el programa imprime el PTR y la acción que aplica, generando una traza de PTRs. Las trazas resultantes son secuencias de hechos en Prolog (*e.g.*, $deambular$($Estado$,avanzar)) y componen la entrada a FOSeq, que aprenderá cómo producir secuencias similares aprendiendo gramáticas de cláusulas definidas (GCDs). El proceso se muestra en el Algoritmo 5.1. En las siguientes secciones se describe el algoritmo en detalle.

5.3 Inducción de gramáticas

Dada una traza de predicados, el algoritmo busca los *n-gramas* (*e.g.*, sub-secuencias de *n*-elementos) que aparezcan al menos dos veces en la secuencia. Como en Apriori [Agrawal y Srikant, 1994], los *n-gramas* candidatos son buscados de forma incremental por su longitud. El *n-grama* con mayor frecuencia es seleccionado, generando una nueva regla para la gramática y reemplazando con un símbolo no-terminal todas las ocurrencias del *n-grama* que aparecen en la secuencia original (ver Algoritmo 5.2).

Algoritmo 5.1 Algoritmo general

Entrada: Un conjunto de secuencias $T = t_1, t_2, \ldots, t_n$
Salida: Una gramática G_{gen} generalizada

1: **for** $i = 1$ to n **do** {Aprender una gramática para cada secuencia}
2: $g_i = inducegram(t_i)$
3: **end for**
4: **for** $i = 1$ to n **do**
5: **if** $elemdif(g_i)$ existe **then** {Si la gramática tiene elementos diferentes a la gramática mejor evaluada}
6: $G_{gen} = gen(G_{max}, g_i)$ {Obtener una gramática general usando la gramática mejor evaluada y las gramáticas que tengan elementos diferentes}
7: **end if**
8: **end for**
9: **Return** G_{gen}

Algoritmo 5.2 Inducción de gramáticas

Entrada: Una secuencia S
Variables locales: índice c para cada nueva regla
Salida: Una gramática R

1: $c = 0$, $R = \emptyset$
2: **while** S tiene elementos repetidos **do**
3: Encontrar los n-gramas con frecuencia ≥ 2 y hacer una lista de candidatos
4: Seleccionar el n-grama con la frecuencia máxima
5: **if** existe más de uno **then**
6: seleccionar el de mayor longitud
7: **else if** la frecuencia de los n-gramas es la misma **then**
8: seleccionar aleatoriamente un n-grama
9: **end if**
10: Sea r_c un nuevo símbolo no-terminal
11: $R \leftarrow R \cup \{r_c \rightarrow n\text{-grama}\}$
12: Reemplazar cada ocurrencia de r_c en S
13: $c = c + 1$
14: **end while**
15: **Return** R

Ilustraremos el proceso de inducción de gramáticas con la siguiente secuencia de constantes: $S \rightarrow$ `a b c b c b c b a b c b d b e b c` (ver Tabla 5.2).

FOSeq busca *n-gramas* con frecuencia ≥ 2 como candidatos para construir una regla. En la primer iteración el candidato con mayor frecuencia es {`b c`} con cinco apariciones en la secuencia por lo se convierte en el cuerpo de la nueva regla $R1 \rightarrow$`b c`. Este *n-grama*

$S \rightarrow$ a b c b c b c b a b c b d b e b c
b c **(5)**
Agrega: $R1 \rightarrow$ b c
$S_1 \rightarrow$ a $R1$ $R1$ $R1$ b a $R1$ b d b e $R1$ Eliminando elementos repetidos: $S_1 \rightarrow$ a $R1$ b a $R1$ b d b e $R1$
$R1$ b (2) a $R1$ (2) a $R1$ b **(2)**
Agrega: $R2 \rightarrow$ a $R1$ b
$S_2 \rightarrow R2$ $R2$ d b e $R1$ Eliminando elementos repetidos: $S_2 \rightarrow R2$ d b e $R1$

Tabla 5.2: Ejemplo de inducción de gramáticas. Los elementos repetidos se eliminan ya que ésto significa que una sub-secuencia se repite en la traza. Sin embargo, nuestras reglas tienen la información de los estados para ser aplicadas, por lo que eliminar elementos repetidos no afecta el desempeño del sistema. Si el estado es tal que la regla es aplicable nuevamente, esta regla es usada.

es reemplazado por el símbolo no-terminal $R1$ en la secuencia S, produciendo S_1 y eliminando elementos repetidos. En nuestro caso, tener elementos repetidos significa que una sub-secuencia se repite en la traza. Sin embargo, nuestras reglas tienen la información de los estados para ser aplicadas, por lo que eliminar elementos repetidos no afecta el desempeño del sistema. Si el estado es tal que la regla es aplicable nuevamente, esta regla es usada. En la segunda iteración FOSeq encuentra tres candidatos: {$R1$ b}, {a $R1$} y {a $R1$ b} con dos repeticiones cada uno. FOSeq selecciona el elemento de mayor longitud: {a $R1$ b} y una nueva regla es agregada: $R2 \rightarrow$a $R1$ b. La secuencia S_2 no tiene elementos repetidos y el proceso termina. La gramática inducida genera una secuencia sin repeticiones S_2, no genera la secuencia original S. La Figura 5.1 muestra la gramática inducida donde $R1$ y $R2$ pueden verse como sub-conceptos en la secuencia.

$S_2 \rightarrow R2$ d b e $R1$
$R1 \rightarrow$ b c
$R2 \rightarrow$ a $R1$ b

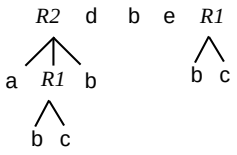


Figura 5.1: Gramática inducida para la secuencia S. S_2 es la secuencia comprimida después del proceso de inducción. La gramática describe la estructura jerárquica de la secuencia, donde $R1$ y $R2$ son reglas que pueden verse como sub-conceptos.

Cuando los elementos de la secuencia son predicados de primer orden, la gramática inducida es una GCD. Las GCDs son una extensión de las gramáticas libres de contexto que son expresadas y ejecutadas en Prolog. En esta tesis tenemos secuencias de predicados que son pares estado-acción como los siguientes:

$$pred1(Estado1,\texttt{acción}_1), pred2(Estado2,\texttt{acción}_1), pred1(Estado3,\texttt{acción}_2), \ldots$$

Para predicados repetidos se crea un nuevo predicado, donde *Estado* es el estado del primer predicado y *Acción* es la acción del último predicado. Por ejemplo, suponiendo que el siguiente par de predicados se repite varias veces en la secuencia:

$$\ldots, pred1(Estadoj,\texttt{acción}_1), pred2(Estadok,\texttt{acción}_2), \ldots$$

se crea el siguiente predicado:

$$nuevopred(Estado1,\texttt{acción}_2) \leftarrow pred1(Estado1,\texttt{acción}_1), pred2(Estado2,\texttt{acción}_2).$$

Una vez inducido el conjunto de gramáticas a partir de las secuencias se realiza la evaluación de las gramáticas inducidas como se describe en la siguiente sección.

5.4 Evaluación de gramáticas

Para cada secuencia se induce una gramática. Cada gramática aprendida se usa para analizar sintácticamente todas las secuencias del conjunto proporcionado por el usuario. La gramática mejor evaluada es seleccionada (ver Algoritmo 5.3). Por cada secuencia analizada, la gramática recibe una evaluación, calculada por la función 5.1.

$$eval(G_i) = \sum_{j=1}^{n} \frac{c_j}{c_j + f_j} \tag{5.1}$$

donde c_j y f_j representan el número de elementos que la gramática es capaz o incapaz de aceptar, respectivamente. Cuando una gramática no puede aceptar un símbolo, ya sea por desconocerlo o por no estar en el orden correcto, el símbolo se descarta y es contado como error, continuando con el análisis de los elementos restantes. La evaluación se realiza después de aprender la gramática para cada secuencia de acuerdo al algoritmo 5.2. Se obtiene

Algoritmo 5.3 Evaluación de gramáticas

Entrada: Un conjunto de secuencias S y un conjunto de gramáticas G correspondiente a cada secuencia de S

Variables locales: i,j,c,f

Salida: La gramática G_{max} mejor evaluada

1: **for** $i = 1$ to número de gramáticas **do**
2: **for** $j = 1$ to número de secuencias n **do** {analiza la secuencia con cada gramática}
3: calcular $eval\_una(g_i, s_j) = \frac{c_j}{c_j + f_j}$
4: $suma\_eval(g_i) = suma\_eval(g_i) + eval\_una(g_i, s_j)$
5: **end for**
6: $eval(g_i) = \frac{suma\_evalg(g_i)}{n}$
7: **end for**
8: $eval_{max}(g_{max}) = max(suma\_eval(g_1), ..., suma\_eval(g_n))$
9: **Return** g_{max}

un conjunto de gramáticas de las cuales FOSeq selecciona la que haya obtenido mejor evaluación, indicando que es la que cubre mejor al conjunto de secuencias.

Ejemplo: Supongamos que un personaje de un juego tiene que buscar un objeto y para ello cuenta con un conjunto de movimientos y acciones. El estado está dado por sensores que registran las distancias hacia los objetos. El objetivo es aprender una estrategia simple para que el personaje logre su objetivo. Para éllo un usuario guía al personaje y se registran las siguientes secuencias:

Secuencia-1
$mover(Estado$,`avanzar`$), buscar(Estado$,`derecha`$), mover(Estado$,`avanzar`$),$ $buscar(Estado$,`derecha`$)$

Secuencia-2
$mover(Estado$,`avanzar`$), buscar(Estado$,izquierda$), mover(Estado$,`avanzar`$)$ $buscar(Estado$,`izquierda`$), mover(Estado$,`avanzar`$), buscar(Estado$,`izquierda`$)$

Secuencia-3
$mover(Estado$,`avanzar`$), buscar(Estado$,`salta`$), mover(Estado$,`avanzar`$)$ $buscar(Estado$,`izquierda`$), mover(Estado$,`avanzar`$), buscar(Estado$,`saltar`$)$

Las secuencias 1, 2 y 3 muestran al personaje realizando la búsqueda de tres formas dis-

tintas, avanzando y buscando a la derecha, avanzando y buscando a la izquierda y buscando y saltando. De estas secuencias se inducen las siguientes gramáticas:

Gramática-1

$S \rightarrow R1$

$R1 \rightarrow mover(Estado,\texttt{avanzar}), buscar(Estado,\texttt{derecha})$

Gramática-2

$S \rightarrow R1$

$R1 \rightarrow mover(Estado,\texttt{avanzar}), busca(Estado,\texttt{izquierda})$

Gramática-3

$S \rightarrow R1, mover(Estado,\texttt{avanzar}), buscar(Estado,\texttt{izquierda}), R1$

$R1 \rightarrow mover(Estado,\texttt{avanzar}), buscar(Estado,\texttt{saltar})$

Una vez aprendidas las gramáticas, se evalúa cada una analizando sintácticamente el conjunto de secuencias. Las evaluaciones para las gramáticas se muestran en la Tabla 5.3.

	No. elementos	Aceptados	No aceptados	Evaluación
Gramática-1				
Secuencia-1	4	4	0	100.00 %
Secuencia-2	6	3	3	50.00 %
Secuencia-3	6	3	3	50.00 %
Promedio				66.67 %
Gramática-2				
Secuencia-1	4	2	2	50.00 %
Secuencia-2	6	6	0	100.00 %
Secuencia-3	6	4	2	66.67 %
Promedio				72.33 %
Gramática-3				
Secuencia-1	4	2	2	50.00 %
Secuencia-2	6	5	1	83.33 %
Secuencia-3	6	6	0	100.00 %
Promedio				**77.67** %

Tabla 5.3: Evaluación de las gramáticas

La primer columna corresponde a las secuencias analizadas (*i.e.*, Secuencia-1, Secuencia-2 y Secuencia-3). La segunda columna indica el número de elementos (predicados) que tiene

cada secuencia. Las siguientes dos columnas indican el número de elementos aceptados y no aceptados por la gramática evaluada. La columna "Evaluación" muestra el porcentaje de la secuencia aceptada por la gramática. El último renglón muestra el promedio de los elementos aceptados por la gramática en el conjunto de secuencias analizadas.

La Tabla 5.3 muestra que la gramática mejor evaluada es la Gramática-3, obteniendo un porcentaje promedio de 77.67 %. Una vez seleccionada la mejor gramática, el siguiente paso es mejorar su cobertura mediante un proceso de generalización que es descrito en la siguiente sección.

5.5 Generalización

La idea básica del proceso de generalización es obtener una nueva gramática que mejore la cobertura de la gramática seleccionada. Por ejemplo, si la mejor gramática describe una trayectoria de navegación de un robot móvil cuando la meta se encuentra a su izquierda, se esperaría que otra secuencia proporcionara información sobre cómo llegar a una meta ubicada a la derecha del robot. La generalización producirá una cláusula que cubra ambos casos mediante la generalización menos general (*lgg*) (ver Algoritmo 5.4) de dos cláusulas [Plotkin, 1969]. El proceso consta de los siguientes pasos:

1. Seleccionar la mejor gramática g_{max}.

2. Seleccionar la gramática g_{otra} que proporciona el mayor número de diferentes unificaciones de predicados. En la gramática pueden encontrarse predicados con dife-

Algoritmo 5.4 Algoritmo de *lgg* entre dos términos

Entrada: Dos términos o literales compatibles L_1 y L_2
Variables locales: P_1,P_2,t_1,t_2,X
Salida: $lgg(L_1, L_2)$
1: $P_1 = L_1$
2: $P_2 = L_2$
3: Encontrar dos términos, t_1 y t_2, en el mismo lugar en P_1 y P_2, tal que $t_1 \neq t_2$ o bien t_1 y t_2 tienen un nombre de función diferente o por lo menos t_1 o t_2 es una variable.
4: Si no existe ese par t_1 y t_2, entonces terminar y $P_1 = P_2 = lgg(L_1,L_2)$.
5: Si existe, escoger una variable X distinta de cualquier variable que ocurra en P_1 o P_2, y en donde t_1 y t_2 aparezcan en el mismo lugar en P_1 y P_2, reemplazarlos con X.
6: Ir a 2.
7: **Return:** $lgg(L_1, L_2)$

rentes constantes, por ejemplo, si la gramática-1 presenta únicamente el predicado $forma(Estado,\texttt{vertical})$ y la gramática-2 presenta en sus reglas el mismo predicado con diferentes constantes como: $forma(Estado,\texttt{vertical})$ y $forma(Estado,\texttt{horizontal})$, entonces la gramática 2 es seleccionada.

3. Calcular el *lgg* entre reglas de g_{max} y g_{otra} con diferentes unificaciones de predicados y reemplaza la regla de g_{max} con el resultado de la regla generalizada.

4. Si la nueva regla mejora la cobertura original, es aceptada, si no, es descartada.

5. El proceso termina hasta alcanzar un umbral de cobertura, las reglas de g_{max} cubren todas las reglas en las otras gramáticas, o hasta que no hay mejoras en el proceso de generalización.

Las trazas representan diferentes unificaciones de un problema del cual pueden inducirse distintas gramáticas. El proceso de generalización se usa para generar una gramática más general aplicable a diferentes trazas del problema. Por ejemplo, dado el siguiente par de cláusulas de dos reglas de una gramática:

$$\begin{aligned} \mathrm{c}_1 = pred(Estado,\texttt{acción}_2) \leftarrow \\ cond1(Estado,\texttt{acción}_1), \\ cond2(Estado,\texttt{acción}_2). \end{aligned} \qquad \begin{aligned} \mathrm{c}_2 = pred(Estado,\texttt{acción}_3) \leftarrow \\ cond1(Estado,\texttt{acción}_1), \\ cond2(Estado,\texttt{acción}_3). \end{aligned}$$

Calculando el $lgg(\mathtt{c}_1,\mathtt{c}_2)$, la cláusula produce:

$$\begin{aligned} pred(Estado, Acción) \leftarrow \\ cond1(Estado,\texttt{acción}_1), \\ cond2(Estado, Acción). \end{aligned}$$

donde las constantes $\texttt{acción}_2$ y $\texttt{acción}_3$ son reemplazadas por la variable *Acción* (ver Algoritmo 5.5). La siguiente sección describe cómo aprender la tarea de navegación a partir de secuencias de PTRs básicos de navegación.

Continuando con el ejemplo de la sección anterior, a partir de la gramática mejor evaluada (Gramática-3) se aplicará el proceso de generalización.

1. La gramática mejor evaluada se une a la que aporta más unificaciones diferentes en una lista de reglas L. Siguiendo con el ejemplo, las Gramática-1 y 2 aportan una nueva

Algoritmo 5.5 Algoritmo de generalización

Entrada: Gramática mejor evaluada g_{max}, lista ordenada L de gramáticas de acuerdo al número de unificaciones diferentes
Variables locales: g_{unida}, c_1 **y** c_2
Salida: Gramática generalizada g_{gen}

```
1: while existen gramáticas con elementos diferentes do
2:    Unir la mejor gramática y la primera de la lista L en g_unida
3:    while existen reglas en g_unida que pueden ser generalizadas do
4:       c_gen = lgg(c_1, c_2) {donde c_1 y c_2 son la primera y segunda regla de la lista respectivamente}
5:       Agregar c_gen a la nueva gramática g_gen
6:       Eliminar c_1 y c_2 de g_unida
7:    end while
8: end while
9: Return gramática generalizada g_gen
```

unificación cada una. $R1$ es la regla de la mejor gramática y $T1$ es la regla de la Gramática-1

$$R1 \rightarrow mover(Estado,\texttt{avanzar}), buscar(Estado,\texttt{saltar})$$
$$T1 \rightarrow mover(Estado,\texttt{avanzar}), buscar(Estado,\texttt{izquierda})$$

2. El número de literales es igual. El nombre del predicado y los argumentos de la primer literal son iguales por lo que se conserva `avanzar`. El nombre del predicado de la segunda literal es igual y los argumentos son constantes por lo que se sustituyen por la variable *Acción*. Se crea una nueva regla $Z1$.

3. Se eliminan $R1$ y $T1$ de L.

4. $Z1$ se agrega a la gramática generalizada g_{gen}.

$$Z1 \rightarrow mover(Estado,\texttt{avanzar}), buscar(Estado, Acción)$$

5. Se une $Z1$ con la siguiente gramática que tiene instanciaciones diferentes.

$$Z1 \rightarrow mover(Estado,\texttt{avanzar}), busca(Estado, Acción)$$
$$R1 \rightarrow mover(Estado,\texttt{avanzar}), busca(Estado,\texttt{derecha})$$

Se crea una nueva regla $Z2$ que sustituye la constante `derecha` por la variable *Acción* en el predicado `buscar`.

$Z2 \rightarrow mover(Estado,\texttt{avanzar}), buscar(Estado, Acción)$

Se elimina $Z1$ y $R1$ de la lista L y se agrega $Z2$ a g_{gen}

6. Como ya no existen reglas para generalizar el proceso termina. La gramática generalizada es g_{gen}.

El resultado es una regla a la que se le puede nombrar $estrategia1$ que consiste en avanzar y luego buscar el objeto ejecutando cualquiera de las posibles acciones. El proceso de generalización origina que en vez de tener tres reglas indicando las acciones, se tiene sólo una.

$$estrategia1(Estado, Acción) \rightarrow mover(Estado,\texttt{avanzar}), buscar(Estado, Acción).$$

En la siguiente sección se describe cómo se aprende el PTR para ir a un punto.

5.6 Aprendizaje del PTR para *ir a un punto*

Una vez que se han aprendido PTRs básicos, el objetivo es aprender programas que usen esos PTRs. El usuario obtiene trazas de la tarea, éstas son transformadas a secuencias de PTRs básicos que se dan como entrada al algoritmo FOSeq. El resultado es una gramática cuyas reglas son los PTRs que combinan los PTRs básicos.

El usuario proporciona trazas de cómo desplazarse entre dos lugares sin colisiones. Las trazas son transformadas en términos de secuencias de PTRs básicos previamente aprendidas. En este ejemplo usamos sólo ***deambular*** y ***orientarse***. Así, dada una secuencia formada por los PTRs ***deambular*** y ***orientar***. FOSeq se usa para aprender una gramática que pueda conducir al robot entre dos lugares usando esos PTRs y posiblemente inducir conceptos intermedios.

El usuario guió al robot móvil para alcanzar distintos lugares generando 8 trazas. A continuación se muestra una sub-secuencia de una traza. FOSeq indujo 8 gramáticas, una por cada traza. Después de ser evaluadas, la gramática mejor evaluada es la correspondiente a la secuencia 5 (*i.e.*, Gram-5) que cubre 99.29 % del conjunto de secuencias.

$orientar(Estado$,`girar-izquierda`$), deambular(Estado$,`girar-izquierda`$),$
$deambular(Estado$,`girar-derecha`$), deambular(Estado$,`avanzar`$),$
$orientar(Estado$,`girar-izquierda`$), deambular(Estado$,`avanzar`$),$
...
$deambular(Estado$,`girar-izquierda`$), deambular(Estado$,`avanzar`$), en\text{-}meta(Estado$,`nil`$)$

El proceso de generalización toma la gramática mejor evaluada (*i.e.*, Gram-5) y la gramática con el número más alto de unificaciones diferentes de los predicados ***deambular*** y ***orientarse*** (*i.e.*, Gram-4) (ver Tabla 5.4) y calcula el *lgg* entre pares de reglas de gramáticas. El proceso itera hasta que no existen mejoras o la gramática generalizada cubre las reglas de otras gramáticas.

Gramática	Evaluación	Nuevos elementos
Gram-5	99.29	0
Gram-4	99.26	2
Gram-7	99.22	1
Gram-6	99.10	2
Gram-2	99.07	1
Gram-1	99.01	1
Gram-3	98.84	1
Gram-8	98.42	1

Tabla 5.4: Evaluación de gramáticas y nuevos elementos

La Tabla 5.5 muestra las reglas generalizadas aprendidas de una traza de 8 secuencias. Los nombres de predicados fueron dados por el usuario.

Cada regla describe una sub-tarea a lo largo de la trayectoria de navegación. Por ejemplo, $R1$ describe el comportamiento *girando-hacia-meta* (*i.e.*, $orientar(Estado1, Acción)$,

($R0$) $meta\text{-}alcanzada(Estado$,`nil`$)$	$\rightarrow en\text{-}meta(Estado)$
($R1$) $girando\text{-}hacia\text{-}meta(Estado1$,`avanzar`$)$	$\rightarrow orientar(Estado1, Acción), deambular(Estado2$,`avanzar`$)$
($R2$) $apuntando\text{-}a\text{-}meta(Estado1, Acción)$	$\rightarrow orientar(Estado1$,`igual`$), deambular(Estado2, Acción)$
($R3$) $imposible\text{-}orientar(Estado1, Acción)$	$\rightarrow orientar(Estado1$,`ninguna`$), deambular(Estado2, Acción)$
($R4$) $pasear(Estado, Acción)$	$\rightarrow deambular(Estado, Acción)$

Tabla 5.5: Reglas del PTR para ***ir a un punto***

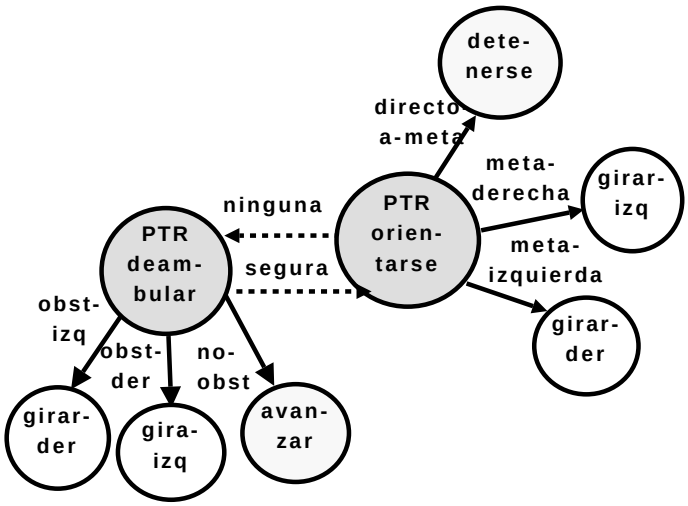


Figura 5.2: Árbol TR para ***ir a un punto***

deambular($Estado2$,`avanzar`)) cuando el robot está girando hacia la izquierda/derecha y avanzando hacia la meta. $R2$ describe el comportamiento *apuntando-a-meta* (*i.e.*, *orientar* ($Estado1$,`igual`), *deambular*($Estado2, Acción$)) cuando el robot no necesita girar porque la meta está en la misma dirección y el robot puede alcanzar la meta. $R3$ representa el comportamiento *imposible-orientar* (*i.e.* *orientar*($Estado1$,`ninguna`), *deambular*($Estado2, Acción$)) cuando el robot no tiene una acción aplicable al estado. $R4$ describe las partes de la trayectoria donde el robot deambuló porque encontró obstáculos o estados desconocidos. $R0$ es la meta agregada por el usuario. La Figura 5.2 muestra el PTR para ir a un punto. Por ejemplo, el robot es capaz de evadir un obstáculo y de regresar para alcanzar su punto destino.

Las jerarquías que se forman dependen de los elementos que constituyen una secuencia de navegación por lo que al agregar nuevos PTRs a la secuencia, la gramática aprendida mostrará nuevas sub-tareas que el robot puede realizar. Las jerarquías representan, como se vió en la Tabla 5.5, diversas sub-tareas encontradas a lo largo de la trayectoria. El aprendizaje de gramáticas para navegación presenta las siguientes ventajas:

- Las gramáticas aprendidas muestran las sub-tareas a lo largo de una trayectoria de navegación.
- Se pueden integrar fácilmente al programa de control de un robot ya que las GCDs son programas en Prolog.

El ejemplo descrito se hizo con secuencias de PTRs ***deambular*** y ***orientarse***. Se apren-

dieron otros PTRs como ***salir de trampa*** y ***seguir*** (capitulo 4) y posteriormente se aprendieron PTRs jerárquicos para ***ir-a*** que incluyen esos PTRs básicos.

La Figura 5.3 muestra el árbol de acciones para el PTR jerárquico que incluye los PTRs básicos: ***orientarse***, ***deambular*** y ***salir de trampa***. El PTR ***deambular*** es una acción para ***salir de trampa*** que su vez tiene como acción al PTR ***orientarse***.

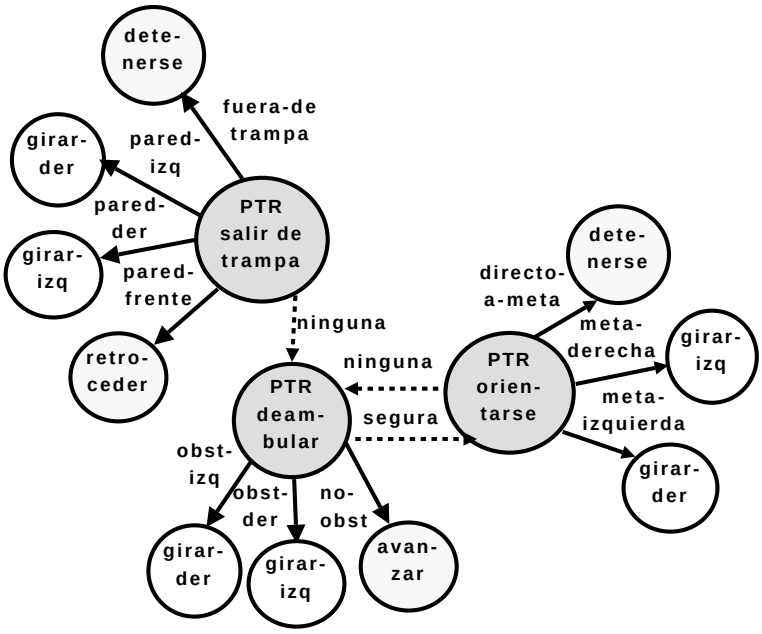


Figura 5.3: Árbol de jerarquías de acciones para ***orientarse***, ***deambular*** y ***salir de trampa***.

La Tabla 5.6 muestra los PTRs que forman las secuencias y el número de secuencias utilizadas para el aprendizaje. Los PTRs apendidos se muestran en el apéndice B.

PTR	No. de secuencias
deambular + *orientarse* (*ir a un punto*)	12
deambular + *orientarse* + *salir de trampa*	16
seguir + *deambular*	10
seguir + *deambular* + *salir de trampa*	14

Tabla 5.6: PTRs jerárquicos aprendidos y número de trazas

5.7 Reconocimiento de ademanes

Los ademanes son una forma de comunicación no-verbal realizada con alguna parte del cuerpo. El uso de ademanes permite a las personas expresar sentimientos, pensamientos y aún comandos para instruir/dirigir a una persona o robot a realizar algo. Proporcionar a un robot la capacidad de reconocer ademanes le facilitaría la integración a un entorno doméstico pues la interacción con las personas sería más parecida a la comunicación humana.

El reconocimiento de ademanes presenta el problema del manejo de grandes cantidades de datos, reconocimiento de ademanes generados por la misma persona y ademanes generados por diferentes personas. En nuestro caso, también nos interesa el reconocimiento de sub-ademanes comunes a diferentes ademanes.

FOSeq transforma información de bajo-nivel de los sensores en una representación relacional. Tenemos secuencias de pares estado-valor, donde un valor puede ser una acción o una variable de verdad como se describe más adelante. Las gramáticas inducidas también representan sub-ademanes comunes a distintos ademanes. En esta sección se describe cómo aprender gramáticas de secuencias de ademanes y cómo se usan para clasificar nuevas secuencias.

Usamos una base de datos[1] de 7,308 muestras de 9 ademanes dinámicos tomados de 10 hombres y 5 mujeres. La Figura 5.4(a) muestra la posición inicial y final para todos los ademanes. El conjunto completo de ademanes se muestra en las Figuras 5.4(b)-(j): `atención`, `acercar`, `izquierda`, `derecha`,`detenerse`, `girar-hacia-la-derecha`, `girar-hacia-la-izquierda`, `saludar` y `apuntar`. A continuación se describen:

1. `atención`: el usuario levanta la mano encima de su rostro (Figura 5.4 (b)).
2. `acercarse`: el usuario mueve la palma de la mano hacia su torso (Figura 5.4 (c)).
3. `izquierda`: el usuario mueve la mano hacia su izquierda (Figura 5.4 (d)).
4. `derecha`: el usuario mueve la mano hacia su derecha (Figura 5.4 (e)).
5. `detener`: el usuario estira el brazo hacia el frente y dirige la palma de la mano hacia la cámara (Figura 5.4 (f)).
6. `girar-hacia-la-derecha`: el usuario con el brazo extendido realiza un movimiento circular en dirección opuesta a las manecillas del reloj (Figura 5.4 (g)).

[1]Base de datos disponible en http://sourceforge.net/projects/visualgestures/

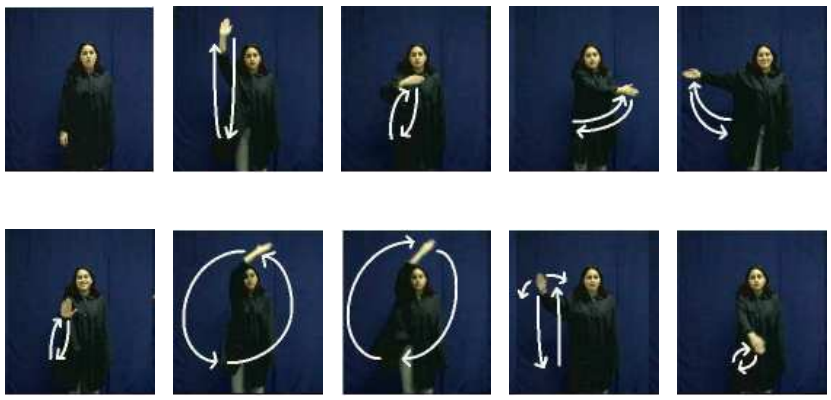


Figura 5.4: Conjunto de ademanes: (a) posición inicial y final, (b) `atención`, (c) `acercar`, (d) `izquierda`, (e) `derecha`, (f) `detener`, (g) `girar-hacia-la-derecha`, (h) `girar-hacia-la-izquierda`, (i) `saludar`, (j) `apuntar`

7. `girar-hacia-la-izquierda`: el usuario con el brazo extendido realiza un movimiento circular en dirección de las manecillas del reloj (Figura 5.4 (h)).

8. `saludar`: el usuario levanta su mano a la altura de su hombro y la balancea de un lado a otro un par de veces (Figura 5.4 (i)).

9. `apuntar`: el usuario estira su brazo y mano sobre su torso señalando algo frente a él (Figura 5.4 (j)).

5.8 Representación de los ademanes

Todos los ademanes son realizados con el brazo derecho e inician y terminan en una posición de "descanso" (Figura 5.4 (a)). De cada ademán, cada persona ejecutó al menos 50 repeticiones y fueron obtenidos usando el sistema visual descrito en [Avilés-Arriaga et al., 2006].

Los ademanes fueron realizados con el brazo derecho y capturados usando el sistema e visión monocular descrito en [Avilés-Arriaga et al., 2006]. Cada secuencia es un vector formado por conjuntos de siete atributos que describen al ademán. Un ejemplo de una secuencia es la siguiente:

$$(++--\ \texttt{T F F}),(++-+\ \texttt{T F F}),(+++0\ \texttt{T F F}),(+++{+}\texttt{T F F})...$$

donde los primeros tres atributos describen movimiento y el resto describen postura. Los atributos de movimiento son $\Delta area$, Δx y Δy, representando cambios en el área de la mano

y cambios en la posición de la mano en el eje X y en el eje Y de la imagen, respectivamente. Estos atributos pueden tomar uno de los valores: $\{+,-,0\}$ indicando incremento, decremento o ningún cambio con respecto al área y posición de la mano en una imagen previa.

Los atributos de postura son *forma, arriba, derecha* y *torso*. Éstos describen la apariencia de la mano y las relaciones espaciales entre la mano y la cara/torso. La apariencia de la mano es descrita por *forma*. Los posibles valores para este atributo son: $\{+,-,0\}$: $(+)$ si la mano está en posición vertical, $(-)$ si la mano está en posición horizontal, y (0) si la mano está inclinada hacia la derecha o izquierda sobre el plano XY. Los atributos *arriba*, *derecha* y *torso* indican si la mano está arriba, a la derecha de la cabeza y sobre el torso.

Las secuencias son transformadas a una representación de primer orden reemplazando sus atributos con un predicado. La Tabla 5.7 muestra el nombre para cada atributo, sus valores, los predicados utilizados y los posibles valores. Un ejemplo de una secuencia transformada se muestra a continuación, donde $Estado$ es una secuencia de bajo nivel de la cual se obtienen los predicados de la secuencia:

$$hmov(Estado,\texttt{derecha}), vmov(Estado,\texttt{subiendo}),$$
$$tamaño(Estado,\texttt{inc}), forma(Estado,\texttt{vertical}),$$
$$derecha(Estado,\texttt{si}),en\_rostro(Estado,\texttt{no}), \ldots$$

Atributo	Nombre	Valores de atributos	Predicados	Valores de los argumentos (Valor)
1	Δx	+, −, 0	*hmov(Estado,Valor)*	`derecha`, `izquierda`, `nada`
2	Δy	+, −, 0	*vmov(Estado,Valor)*	`subiendo`, `bajando`, `nada`
3	$\Delta area$	+, −, 0	*tamaño(Estado,Valor)*	`inc`, `dec`, `igual`
4	*forma*	+, −, 0	*forma(Estado,Valor)*	`vertical`, `horizontal`, `inclinada`
5	*derecha*	`si`, `no`	*derecha(Estado,Valor)*	`si`, `no`
6	*arriba*	`si`, `no`	*en_rostro(Estado,Valor)*	`si`, `no`
7	*torso*	`si`, `no`	*sobre_torso(Estado,Valor)*	`si`, `no`

Tabla 5.7: Atributos de ademanes: de atributos a predicados

5.9 Aprendizaje de gramáticas de ademanes

El objetivo de esta aplicación es aprender una gramática para cada ademán enfocándonos en el reconocimiento de ademanes realizados por una persona. Con el fin de realizar una

comparación con otra técnica, los conjuntos de entrenamiento y prueba se generaron con base en los descritos en [Avilés-Arriaga et al., 2006]:

- De 50 secuencias por ademán se seleccionaron aleatoriamente 20 secuencias para inducir las gramáticas. Las 30 secuencias restantes constituyen el conjunto de prueba.
- Del conjunto de prueba se generaron dos sub-conjuntos seleccionando aleatoriamente 2 y 10 secuencias para cada uno. El objetivo es inducir gramáticas con cada sub-conjunto y comparar la precisión de las gramáticas inducidas al incrementar el número de secuencias de entrenamiento.
- Se seleccionó aleatoriamente un conjunto de ejemplos negativos para aprender las gramáticas. Para cada ademán, los ejemplos negativos son secuencias de los ademanes restantes. El conjunto de ejemplos negativos se construyó tomando un ejemplo de cada ademán restante.
- De cada sub-conjunto se aprendió una gramática para cada ademán, obteniendo 9 gramáticas.
- Se presentó cada secuencia del conjunto de pruebas a las gramáticas aprendidas. La gramática que obtuvo la mejor evaluación corresponde al ademán mostrado.
- Se repitió el proceso 10 veces para cada sub-conjunto.

De los conjuntos de secuencias para entrenamiento se aprenden las gramáticas de cada ademán. A continuación se muestra una sub-secuencia del ademán ***apuntar*** en función de los términos mostrados en la Tabla 5.7.

$hmov(Estado,\texttt{izquierda}), vmov(Estado,\texttt{subiendo}), tamaño(Estado,\texttt{dec}), forma(Estado,$
$\texttt{vertical}), derecha(Estado,\texttt{si}), en\_rostro(Estado,\texttt{no}), sobre\_torso(Estado,\texttt{no})$
$hmov(Estado,\texttt{izquierda}, vmov(Estado,\texttt{subiendo}), tamaño(Estado,\texttt{inc}), forma(Estado,$
$\texttt{vertical}, derecha(Estado,\texttt{si}), \ldots$

El inicio de la secuencia nos dice que el brazo se mueve a la izquierda elevándose ($hmov(Estado,\texttt{izquierda}), vmov(Estado,\texttt{subiendo})$), la mano se mueve un poco hacia atrás y está en posición vertical ($tamaño(Estado,\texttt{dec}), forma(Estado,\texttt{vertical})$) y está a la derecha del cuerpo pero no a la altura de la cara ($derecha(Estado,\texttt{si}), en\_rostro(Estado,\texttt{no})$

). Esta descripción muestra el movimiento inicial y la posición de la mano que una persona realiza al apuntar hacia algún objeto al frente. Es difícil que una persona ejecute varias veces el mismo ademán de la misma manera. En ocasiones puede cruzar su brazo sobre el torso y otras veces que mantenga el brazo a la derecha de su cuerpo. La dirección de los movimientos también pueden variar dependiendo de la posición del objeto al que apunte. La idea es que las gramáticas expresen estas variaciones para que sean capaces de reconocer secuencias del ademán.

De las 20 secuencias de entrenamiento, la gramática mejor evaluada obtuvo 82.86 % de las secuencias cubiertas. La Tabla 5.8 muestra las reglas de la mejor gramática del ademán *apuntar*.

$R1 \rightarrow en\_rostro(Estado,\texttt{no}), sobre\_torso(Estado,\texttt{si})$
$R2 \rightarrow derecha(Estado,\texttt{no}),R1$
$R3 \rightarrow forma(Estado,\texttt{horizontal}),R2$
$R4 \rightarrow hmov(Estado,\texttt{izquierda}), vmov(Estado,\texttt{subiendo})$
$R5 \rightarrow tamaño(Estado,\texttt{inc}),R3$
$R6 \rightarrow forma(Estado,\texttt{vertical}), derecha(Estado,\texttt{si})$
$R7 \rightarrow R6,en\_rostro(Estado,\texttt{no}), sobre\_torso(Estado,\texttt{no})$
$R8 \rightarrow hmov(Estado,\texttt{derecha}), vmov(Estado,\texttt{bajando})$
$R9 \rightarrow R4,tamaño(Estado,\texttt{dec}),R7$
$R10 \rightarrow hmov(Estado,\texttt{izquierda}), vmov(Estado,\texttt{bajando}),R5$
$R11 \rightarrow vmov(Estado,\texttt{subiendo}), tamaño(Estado,\texttt{dec}),R3$
$R12 \rightarrow R8,tamaño(Estado,\texttt{inc})$

Tabla 5.8: Gramática mejor evaluada del ademán `apuntar`

Se pueden identificar reglas de posición, de movimiento y aquéllas que combinan ambos términos. Por ejemplo, $R1$ es una regla de posición que dice que la mano no está a la altura del rostro pero sí se encuentra sobre el torso. $R4$ es una regla de movimiento que dice que la mano se desplaza subiendo hacia la izquierda. Por el contrario, $R8$ muestra un movimiento bajando hacia la derecha, que ocurre cuando la persona ya está regresando el brazo a su posición inicial. A continuación se muestra la secuencia S que describe el ademán `apuntar` utilizando las reglas aprendidas.

$S \rightarrow R9\ R4\ R5\ R4\ tamaño(Estado,\texttt{inc})forma(Estado,\texttt{vertical})\ R2$
$hmov(Estado,\texttt{nada})\ R11\ hmov(Estado,\texttt{derecha})vmov(Estado,\texttt{subiendo})\ R5$
$R10\ hmov(Estado,\texttt{derecha})\ R11\ R10\ R12\ forma(Estado,\texttt{inclinada})\ R2\ R8$
$tamaño(Estado,\texttt{dec})\ R6\ R1\ R12\ R7$

El siguiente paso consiste en obtener la lista de gramáticas con unificaciones diferentes a las encontradas en la mejor gramática. Se encontraron dos gramáticas con unificaciones diferentes. Al efectuar el proceso de generalización se obtuvo la gramática que se muestra en la Tabla 5.9.

$G1 \rightarrow en\_rostro(Estado,\texttt{no}),sobre\_torso(Estado,Valor)$
$G2 \rightarrow derecha(Estado,Valor),G1$
$G3 \rightarrow hmov(Estado,Valor1),vmov(Estado,Valor2)$
$G4 \rightarrow forma(Estado,Valor),G2$
$G5 \rightarrow tama\ no(Estado,Valor),G4$
$G6 \rightarrow derecha(Estado,\texttt{no}),G1$
$G7 \rightarrow G3,G5$
$G8 \rightarrow vmov(Estado,Valor),G5$
$G9 \rightarrow forma(Estado,\texttt{horizontal}),G6$
$G10 \rightarrow G7,hmov(Estado,\texttt{izquierda}),G8$
$G11 \rightarrow G3,tamaño(Estado,Valor)$
$G12 \rightarrow G8,vmov(Estado,\texttt{subiendo}),tamaño(Estado,\texttt{dec}),G9$
$G13 \rightarrow G2,G6$
$G14 \rightarrow tamaño(Estado,Valor),G9$

Tabla 5.9: Gramática generalizada del ademán `apuntar`. Después del proceso de generalización, diversas constantes o atributos son sustituidos por variables. Por ejemplo, la variable Valor en $G1$ acepta los atributos si/no para expresar si la mano está o no sobre el torso

Las reglas de la gramática generalizada muestran la variable Valor en los términos en los cuales puede instanciarse cualquier valor de los indicados en la Tabla 5.7. Por ejemplo, las reglas $R1$ y $G1$ describen la posición de la mano cuando no está a la altura del rostro pero puede o no estar sobre el torso. La variable $Valor$ en $G1$ acepta los valores si/no para expresar si la mano está o no sobre el torso. Ésto significa que existen secuencias en las que la persona ejecutó el ademán `apuntar` cruzando el brazo sobre su torso y otras veces sin cruzarlo, de forma frontal.

Las reglas $R4$ y $G3$ se refieren al movimiento. $G3$ acepta movimiento horizontal a la derecha y a la izquierda y movimiento vertical hacia arriba y hacia abajo. Ésto indica que ambos tipos de desplazamiento (subiendo hacia la izquierda y bajando hacia la derecha) son válidos para este ademán.

Cada regla describe un sub-ademán a lo largo de la secuencia. Durante la secuencia de ejecución de un ademán pueden identificarse partes o sub-ademanes, por ejemplo, cuando el brazo va subiendo, cuando está arriba (en el caso del ademán saludar), cuando baja, etc. La regla $G3$ puede representar el sub-ademán $subir\_bajar(Estado) \rightarrow hmov(Estado,Valor1)$

vmov(Estado,Valor2). El usuario puede sustituir los nombres de las reglas por nombres de predicados alusivos al sub-ademán que representen. El aprendizaje de gramáticas relacionales para reconocimiento de ademanes produce una representación explícita que permite identificar similitudes entre ademanes. En el capítulo 6 se muestran resultados de clasificación usando las gramáticas aprendidas y las similitudes entre ademanes que se observan en los resultados.

El aprendizaje de gramáticas para clasificación de ademanes presenta las siguientes ventajas:

- Proporciona una representación explícita que permite analizar la estructura del ademán.
- Muestran los posibles sub-ademanes en la secuencia que describe al ademán.

5.10 Resumen

En este capítulo se presentó FOSeq, un algoritmo para aprendizaje relacional de gramáticas. FOSeq aprende gramáticas a partir de secuencias de acciones. Las secuencias son dadas como trazas de bajo nivel de datos de sensores que son transformados en una representación de alto nivel.

FOSeq induce una gramática para cada secuencia, evalúa las gramáticas, obtiene una lista de las gramáticas que tienen unificaciones distintas y realiza un proceso de generalización sobre la mejor gramática para cubrir la mayoría de las secuencias.

A diferencia de los enfoques existentes, las gramáticas inducidas por FOSeq pueden ser usadas para ejecutar una tarea particular o para clasificar nuevas secuencias. El enfoque se usó en dos dominios: (i) navegación y (ii) reconocimiento de ademanes.

Se mostró cómo se aprendió el PTR para ir a un punto a partir de secuencias de predicados de trayectorias de un robot móvil y se listan los PTRs jerárquicos para navegación aprendidos. Las reglas describen de forma explícita sub-tareas que se ejecutan a lo largo de la trayectoria.

Para el aprendizaje de ademanes se usó una base de datos pública que incluye secuencias para 9 ademanes. Se describe la representación de alto nivel que se utilizó para las secuencias y se muestra un ejemplo de la gramática aprendida para el ademán “apuntar”. Las reglas muestran de forma explícita sub-ademanes que la persona ejecuta como parte del ademán.

El siguiente capítulo muestra los experimentos usando los PTRs para navegación aprendidos y los resultados de clasificación en nuevas secuencias de ademanes.

6
Experimentos

El objetivo de este capítulo es presentar nuestro trabajo experimental y los resultados obtenidos. Se probaron los PTRs para navegación aprendidos: ***deambular***, ***orientarse***, ***salir de una trampa***, ***seguir*** a un objeto móvil e ***ir a un punto***. Los PTRs para navegación se probaron en simulación y en un robot de servicio Peoplebot. Se muestran resultados de precisión en la ejecución de los PTRs. También se realizaron experimentos para reconocimiento de ademanes utilizando el aprendizaje de gramáticas y se muestran los resultados obtenidos para la clasificación de nueve ademanes de una base de datos pública.

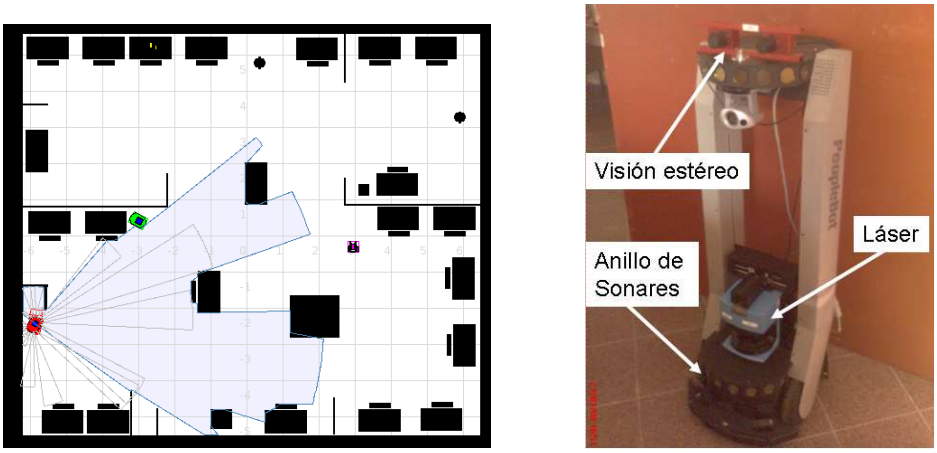
6.1 Experimentos para navegación: plataforma experimental

Los experimentos fueron realizados en simulación (ver Figura 6.1(a)) y con un robot PeopleBot de ActivMedia equipado con un anillo de sonares y un sensor LASER SICK LMS200 (ver Figura 6.1(b)). Se utilizó el software para control de robots *Player/Stage* [Vaughan et al., 2003]. En [Vargas y Morales, 2009] y [Vargas y Morales, 2008] se describe el aprendizaje y resultados de algunos experimentos para navegación.

6.1.1 Objetivos

Los experimentos realizados tienen los siguientes objetivos:

- Probar los PTRs de navegación básicos y jerárquicos en simulación y en el robot real. Mostrar que los PTRs pueden controlar al robot en ambientes diferentes al de entrenamiento. Como ambientes diferentes se consideran mapas en los que nunca se en-



(a) Simulador (b) Robot real

Figura 6.1: Plataformas de experimentos

trenó aunque mantienen características similares ya que corresponden a ambientes de oficinas.

- Mostrar la flexibilidad de los PTRs al poder probarse de forma aislada (*e.g.*, uno a la vez) o combinados dependiendo de las habilidades que el robot necesite.
- Integrar los PTRs a una plataforma de robot de servicio. Los PTRs se probaron como módulo de navegación para un robot móvil que ejecuta tareas de servicio (*e.g.*, mensajería, búsqueda de objetos, seguimiento de personas).
- Comparar el desempeño de los PTRs con un algoritmo deliberativo. La idea es observar ventajas y desventajas entre un enfoque que planea sus trayectorias y uno reactivo como el mostrado en la tesis para construir un enfoque que combine sus características.
- Obtener resultados cuantitativos en la ejecución de las tareas/sub-tareas de navegación.
- Probar las gramáticas de ademanes aprendidas como clasificadores de nuevas secuencias. Se muestran los resultados de clasificación utilizando conjuntos de secuencias de prueba.

6.1.2 Condiciones

- Evaluamos el desempeño de los PTRs en diferentes escenarios, con obstáculos de diversas formas y tamaños, y con obstáculos estáticos y dinámicos para probar la capacidad

del robot para reaccionar a eventos inesperados sin perder su meta. Se utilizaron 10 mapas diferentes a aquéllos en los cuales se realizó la fase de entrenamiento mostrados en la Figura 6.2. Seis de éllos fueron dibujados en un editor (Figura 6.3 (a)-(f)). El resto son mapas de ambientes reales (Figura 6.3 (g)-(j)) construidos utilizando un módulo de construcción de mapas [Jáquez, 2005]. En cada mapa se muestra su medida aproximada.

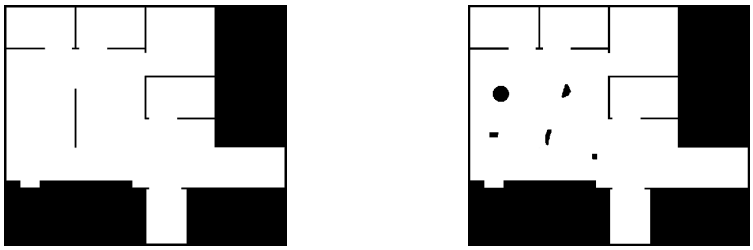
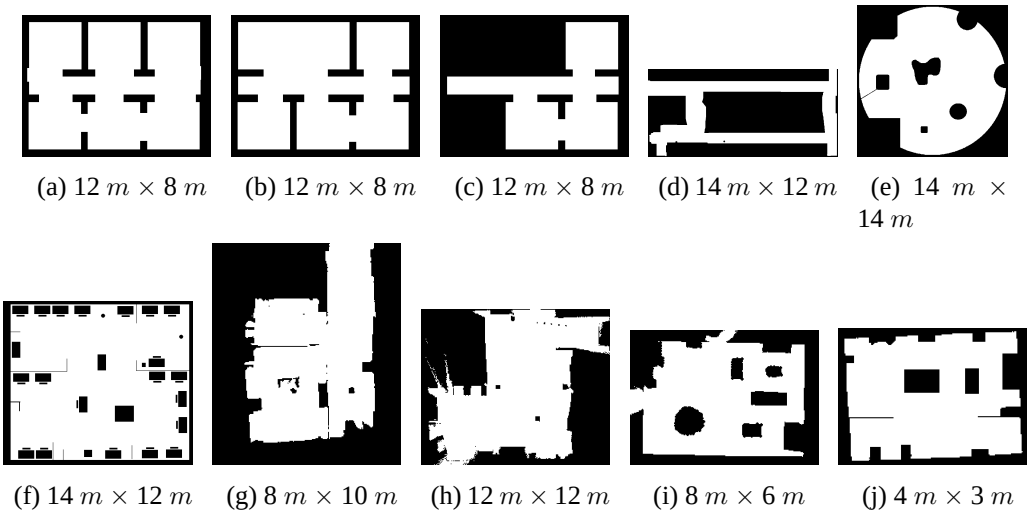


Figura 6.2: Mapas de entrenamiento $14m \times 12m$



(a) 12 $m \times$ 8 m (b) 12 $m \times$ 8 m (c) 12 $m \times$ 8 m (d) 14 $m \times$ 12 m (e) 14 $m \times$ 14 m

(f) 14 $m \times$ 12 m (g) 8 $m \times$ 10 m (h) 12 $m \times$ 12 m (i) 8 $m \times$ 6 m (j) 4 $m \times$ 3 m

Figura 6.3: Mapas usados para los experimentos en simulación: (a)-(f) construidos en un editor de imágenes (g)-(j) obtenidos de un ambiente real

- Los elementos dinámicos en simulación pueden ser de dos tipos: (i) otro robot móvil controlado con *joystick* o con otro PTR y (ii) objetos controlados con *joystick* como los de la Figura 6.4.

- Para preparar los ambientes reales de pruebas se consideraron las limitaciones del

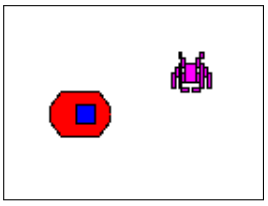


Figura 6.4: Objetos en simulación

sensor láser:

- El rayo sensa aproximadamente a 50 cm sobre el piso de forma paralela. Los objetos que no alcancen esa altura o aquéllos que estén sobre esa medida sin tocar el piso no serán detectados. Por seguridad del robot y del mismo ambiente, los objetos fuera del alcance del sensor se eliminaron o adaptaron para que fueran detectados (*e.g.*, cubriendo espacios huecos en el mobiliario).
- Se evitaron ambientes con paredes de vidrio dado que el sensor no es capaz de detectarlo.

- Los PTRs se evaluaron obteniendo la precisión, definida por el porcentaje de tareas o sub-tareas terminadas exitosamente según se expresa en la fórmula 6.1.

$$p = \frac{e}{e+f} \times 100 \tag{6.1}$$

  Donde e es el número de tareas o sub-tareas realizadas exitosamente y f es el número de tareas o sub-tareas que fallaron.

- La posición inicial del robot y el punto meta (cuando aplica) se generaron aleatoriamente. La localización del robot se toma directamente del simulador. Cada PTR se probó de forma independiente, obteniendo la precisión tanto para los PTRs básicos como para los compuestos.

A continuación se describen los experimentos realizados.

6.2 Simulación

Se realizaron experimentos para el conjunto completo de PTRs aprendidos: (i) PTRs básicos: ***deambular***, ***orientarse***, ***salir de una trampa*** y ***seguir*** a un objeto móvil y (ii)

jerárquicos: ***deambular*/*orientar*** (ir a un punto), ***deambular*/*orientar*/*salir de una trampa***, ***seguir*/*deambular***, ***seguir*/*deambular*/*salir de trampa***. A continuación se describe cómo se probaron los PTRs básicos.

Se realizaron entre 30 y 40 pruebas para cada PTR considerando hacer 3 pruebas por cada mapa (10) y con puntos de inicio y fin seleccionados aleatoriamente. De esta manera, el robot podía ser probado con las dificultades particulares de cada mapa. Se hicieron 40 pruebas para los PTRs que requerían menos tiempo para ejecutarlos.

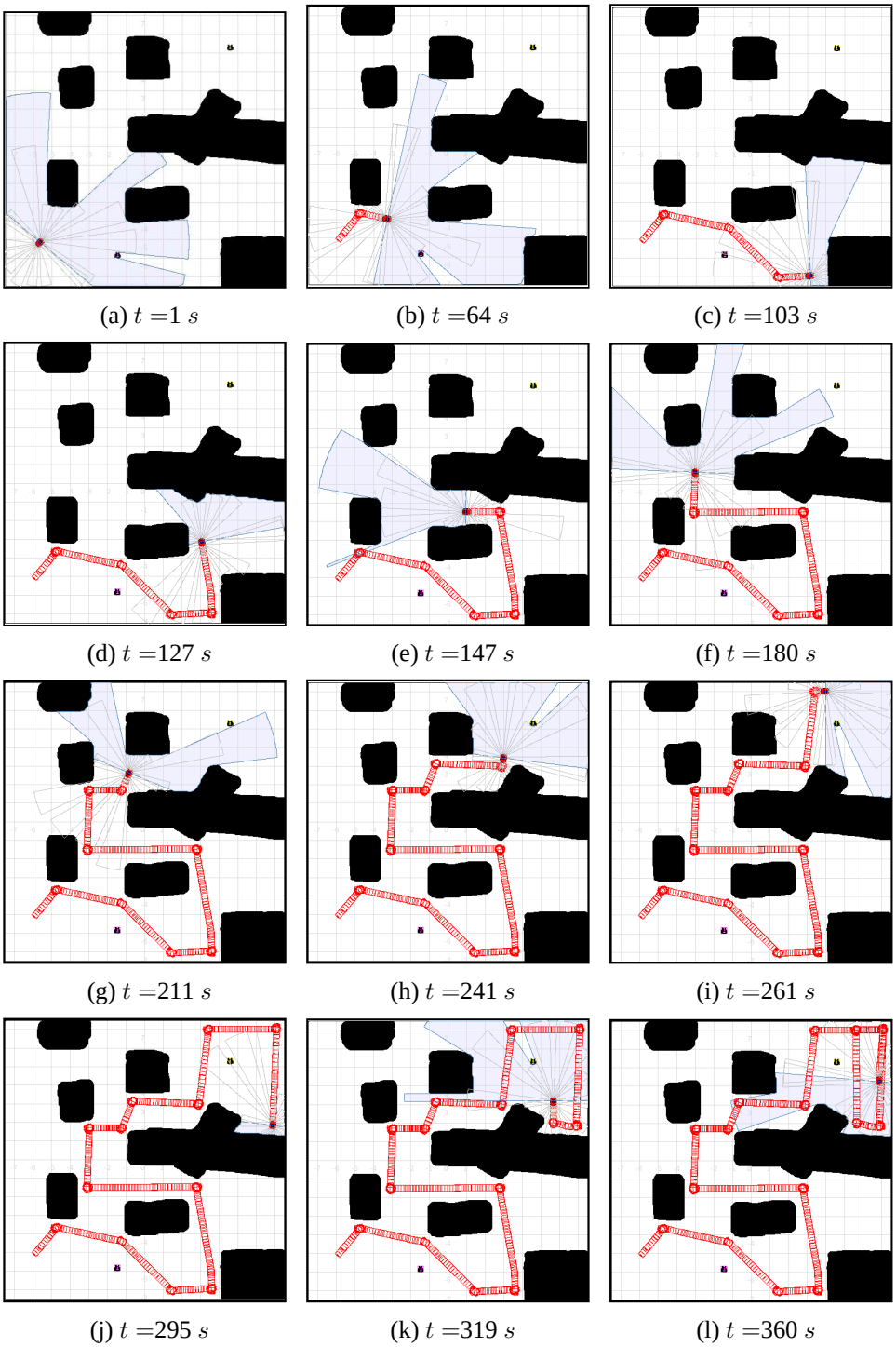
6.2.1 *Deambular*

El objetivo de estos experimentos es que el robot se desplace por distintos ambientes sin chocar con obstáculos fijos y/o móviles. Se realizaron 30 experimentos, 3 en cada mapa, en los cuales el nivel de dificultad se incrementó gradualmente agregando diversos elementos: (i) paredes y pasillos, (ii) obstáculos fijos de distintas formas y tamaños, y (iii) obstáculos dinámicos (*e.g.*, otro robot móvil deambulando). De los 30 experimentos realizados, 20 contaron con objetos dinámicos en el ambiente, los 10 restantes no. Cada experimento tuvo una duración de hasta 30 minutos y se consideró exitoso si el robot no tuvo colisiones. La precisión lograda es de 86.67 %. A continuación se describen dos casos: 1) con obstáculos estáticos de forma irregular, y 2) el robot deambulando en un pasillo.

Caso 1: *Deambular* en un ambiente con obstáculos estáticos de forma irregular

El ambiente en donde el robot tiene que deambular está compuesto por obstáculos estáticos de forma irregular. El robot inicia en el extremo inferior izquierdo (Figura 6.5 (a)) y cuenta con espacio libre al frente por lo que ejecuta la acción `avanzar` hasta que encuentra un obstáculo y tiene que `girar a la derecha` (Figura 6.5 (b)) para evadir el obstáculo y poder `avanzar`. El robot encuentra otro obstáculo más grande y lo evade nuevamente al `girar a la derecha`. Al tener la parte frontal libre, el robot `avanza` hasta que encuentra una pared y `gira a la izquierda`, `avanzando` hasta encontrarse con otro obstáculo (Figura 6.5 (c)). Las Figuras 6.5 (d)-(l) muestran la trayectoria seguida por el robot al deambular. En la Figura 6.5 (l) se observa que el robot sigue el mismo camino ya recorrido. Se muestra el tiempo en segundos en el cual se obtuvo la imagen.

A pesar de que los obstáculos tienen forma diferente a los del ambiente en el cual se realizó el aprendizaje, el robot no tuvo colisiones. Sin embargo, el robot puede quedar atrapado en ciclos, como se observa en la Figura 6.5 (l).



(a) t =1 s (b) t =64 s (c) t =103 s

(d) t =127 s (e) t =147 s (f) t =180 s

(g) t =211 s (h) t =241 s (i) t =261 s

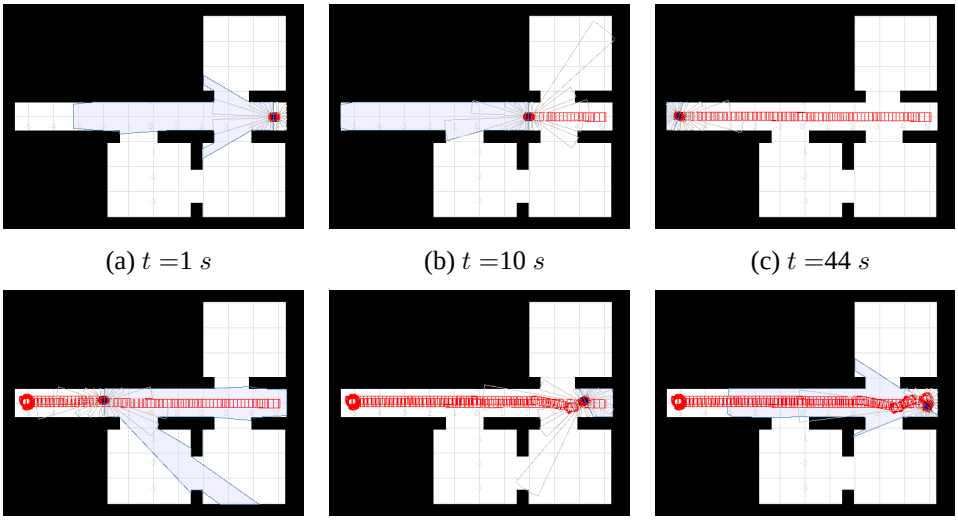
(j) t =295 s (k) t =319 s (l) t =360 s

Figura 6.5: Deambulando con obstáculos estáticos de forma irregular

Caso 2: *Deambular en un pasillo*

Uno de los ambientes más comunes en el interior de una casa u oficina es un pasillo. En esta prueba se espera que el robot se desplace de forma estable, es decir, sin movimientos diagonales que suelen ocurrir en este tipo de ambientes.

El robot inicia en el extremo derecho del pasillo (Figura 6.6 (a)) y `avanza` hasta llegar al otro extremo del pasillo (Figuras 6.6 (a)-(c)). El robot `gira a la derecha` hasta encontrar espacio libre al frente y vuelve a `avanzar` por el pasillo, esta vez de regreso (Figura 6.6 (d)). Cuando el robot pasa muy cerca de una entrada (Figura 6.6 (e)), la considera un obstáculo, por lo que `gira a la izquierda` para evitar chocar. Las Figuras 6.6 (f)-(g) muestran que el robot hizo movimientos de evasión por el resto del pasillo pues ya no hubo espacio suficiente para que avanzara de forma recta.



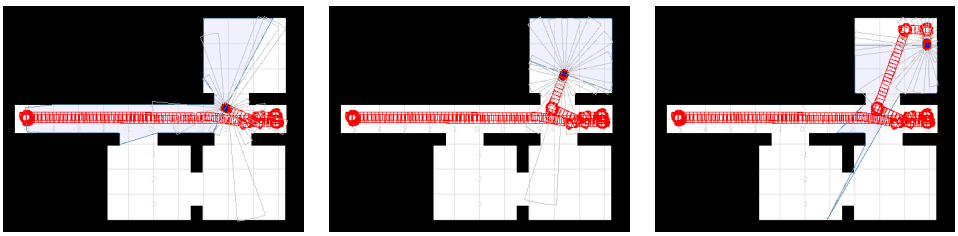
(a) t =1 s (b) t =10 s (c) t =44 s

(d) t =69 s (e) t =106 s (f) t =124 s

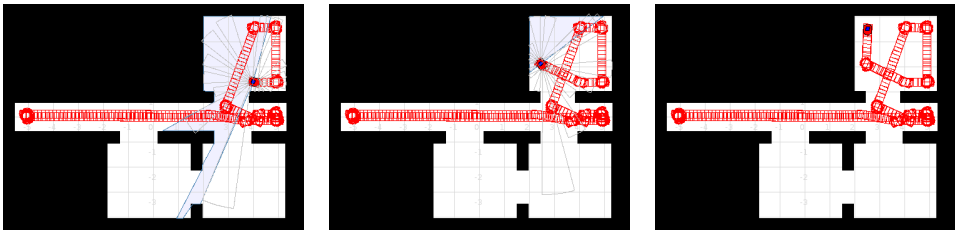
Figura 6.6: ***Deambular*** en un pasillo. Parte 1.

El robot entra en una habitación (Figura 6.7 (h)) y `avanza` hasta encontrar una pared (Figura 6.7 (i)). Para evadir la pared el robot `gira a la derecha` y `avanza` (Figura 6.7 (j)) y continúa deambulando en la habitación (Figura 6.7 (k)-(l)).

El robot logra avanzar por el pasillo sin colisiones y la mayor parte del trayecto lo hace de forma recta. Si el robot encuentra obstáculos, como en este caso, una entrada, el robot efectúa giros para evadirlos y su desplazamiento deja de ser recto. El robot necesita recorrer suficiente distancia para recobrar un comportamiento estable.



(g) t =143 s (h) t =157 s (i) t =188 s



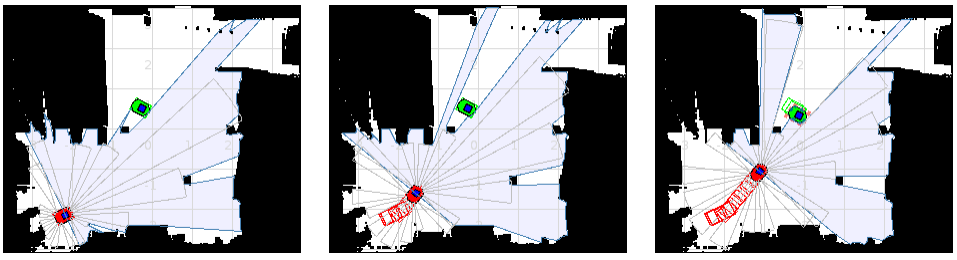
(j) t =201 s (k) t =211 s (l) t =223 s

Figura 6.7: ***Deambular*** en un pasillo. Parte 2.

Caso 3: *Deambular* con otro robot en el ambiente

En este experimento existen dos robots deambulando en el ambiente que deben evitar colisiones tanto con los elementos fijos como entre ellos.

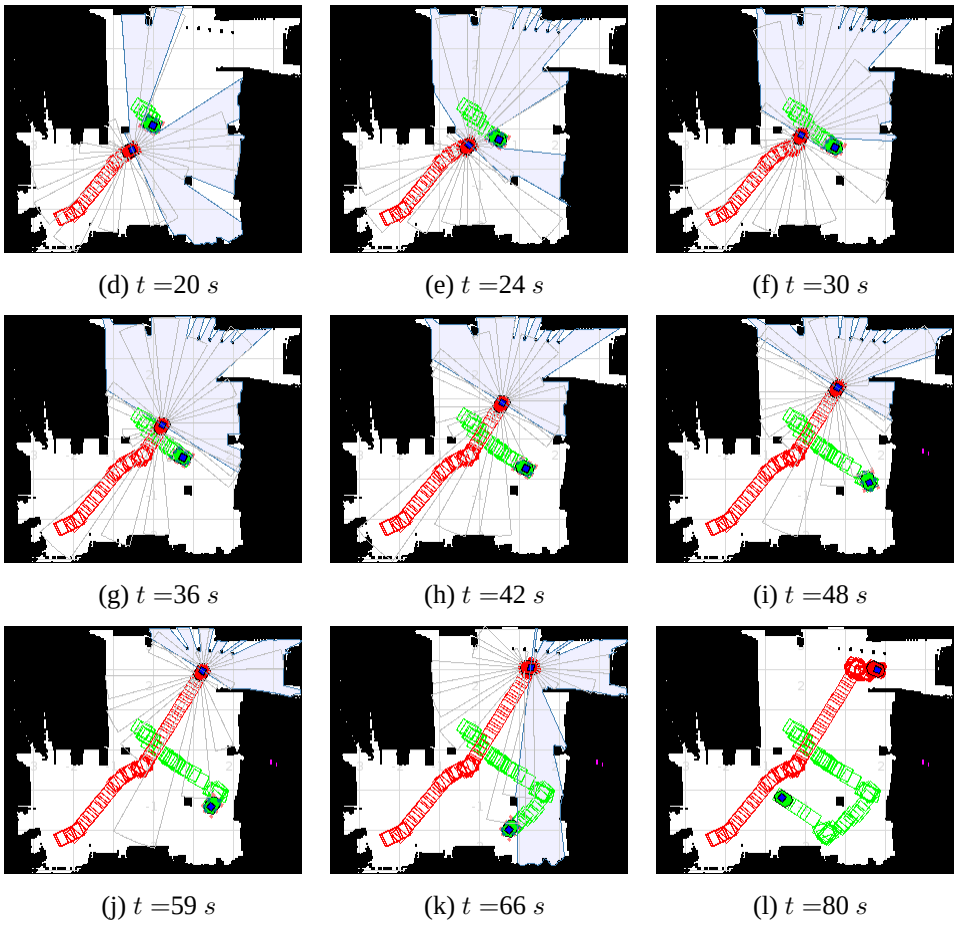
Los robots inician con espacio libre al frente así que `avanzan` (Figura 6.8 (a)-(c)). Los robots se aproximan, uno de ellos, el que baja, percibe que hay espacio libre al frente por lo que su acción es `avanzar` (Figura 6.9 (d)-(f)). El robot que sube, por el contrario, percibe al otro robot como un obstáculo y `gira a la izquierda` para evadirlo (Figura 6.9 (e)-(g)) y ambos continúan deambulando en el ambente (Figura 6.9 (h)-(l)).



(a)t =1 s (b)t =11 s (c) t =16 s

Figura 6.8: Deambulando con otro robot en el ambiente. Parte 1.

Los robots logran detectar y evadir los obstáculos de forma satisfactoria. En general,



(d) t =20 s (e) t =24 s (f) t =30 s

(g) t =36 s (h) t =42 s (i) t =48 s

(j) t =59 s (k) t =66 s (l) t =80 s

Figura 6.9: ***Deambular*** con otro robot en el ambiente. Parte 2.

muestran un comportamiento estable y seguro. Sin embargo, si el objeto dinámico se aproximara con una velocidad más alta, el robot no podría reaccionar lo suficientemente rápido para evitar chocar con el objeto.

6.2.2 *Orientarse*

El objetivo de esta prueba es que, dado un punto meta, el robot sea capaz de identificar una zona adecuada para orientarse y colocarse en dirección a la meta. La dificultad de esta tarea consiste en que si el robot se orienta cuando existe un obstáculo entre él y la meta, puede caer en un ciclo de orientación-evasión del que difícilmente lograría salir. El PTR para orientación debe ser capaz de reconocer esta situación para evitarla. El experimento tiene éxito cada vez que el robot identifica correctamente la zona: si no es una zona de orientación

el robot deambula; si es una zona de orientación, el robot se coloca en dirección a la meta. Se realizaron 30 experimentos, 3 en cada mapa de prueba. Se obtuvo una precisión de 100 %. La Figura 6.10(a) muestra a un robot que inicia reconociendo la zona de orientación para poder colocarse en dirección a la meta. A la zona en donde puede orientarse le llamamos zona de orientación, esta es una zona es libre de obstáculos alrededor del robot. El robot avanza y encuentra un obstáculo rectangular que está entre el y la meta por lo que no puede orientarse en ése lugar. El robot deambula y avanza hasta encontrar nuevamente una zona de orientación. La meta queda entonces a su derecha y puede orientarse directamente hacia ella pues no existen obstáculos entre ambos. Finalmente, el robot llega a la meta. La Figura 6.10(b) muestra al robot en una posición en la que existen obstáculos cercanos entre él y la meta por lo que identifica que no es una zona de orientación. El robot deambula y se orienta repetidamente en su desplazamiento hasta que llega a la meta.

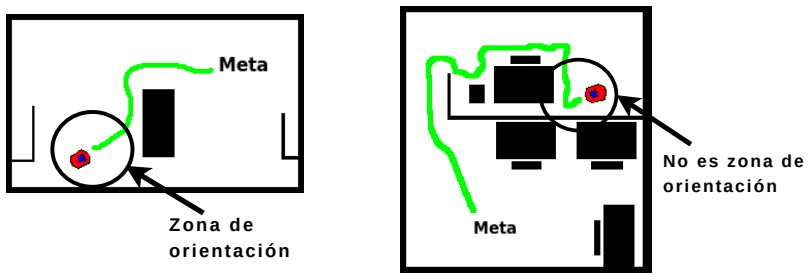


Figura 6.10: En (a) el robot se encuentra en una zona de orientación y puede colocarse en dirección a la meta. En (b) si el robot se orienta quedaría atrapado y debe encontrar una zona apropiada. En este caso la zona de orientación no está libre.

En espacios abiertos el robot no presenta dificultades para lograr orientarse. En contraste, si el espacio es reducido o aglomerado, es posible que el robot necesite deambular repetitivamente y tardar de 5 a 10 minutos de los 30 minutos establecidos para las pruebas hasta lograr encontrar una zona adecuada para orientarse. El tiempo depende del mapa y de la distancia hacia la meta, estamos considerando únicamente los mapas de prueba por lo que estos tiempos pueden variar para otros mapas.

6.2.3 *Salir de trampa*

En estos experimentos, el robot inicia dentro de una trampa en distintas posiciones y a distintas distancias desde la entrada de la trampa (ver Figuras 6.11 (a) y Figuras 6.11 (b)).

El objetivo es que el PTR pueda controlar al robot para que salga de la trampa. Si el robot no logra salir, se considera fallido. Se realizaron 30 pruebas, 3 en cada uno de los mapas de prueba y se logró una precisión del 80 %.

Para mostrar la importancia de este PTR, la Figura 6.11 presenta un ejemplo de un robot que no tiene incorporado este PTR. En la Figura 6.11(a) el robot se orienta hacia la meta pues percibe frente a él, suficiente espacio para avanzar. Sin embargo, no reconoce que está dentro de una trampa. El robot avanza (Figura 6.11(b)) y trata de evadir la pared girando hacia la derecha, como se muestra en la Figura 6.11(c). Finalmente el robot queda atrapado 6.11(d) pues no hay suficiente espacio para que realice otros movimientos.

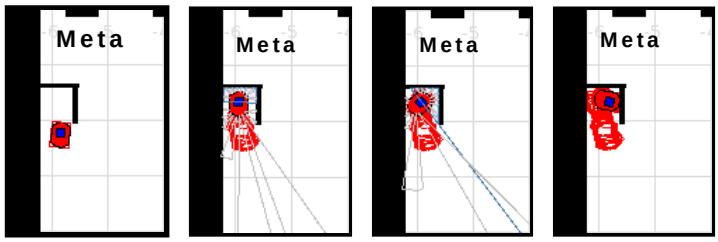


Figura 6.11: Robot que no usa el PTR ***salir de trampa***. El robot: (a) se orienta hacia la meta, (b) entra en la trampa, (c) trata de evadir la pared frontal pero no hay suficiente espacio, y (d) queda atrapado.

Cuando el robot cuenta con este PTR, su comportamiento es diferente. La Figura 6.12 (a) muestra al robot en la misma posición inicial que en el ejemplo anterior pero ahora, si es capaz de identificar la trampa. De esta manera, empieza a girar a la derecha para salir de la trampa (Figura 6.12 (b)).

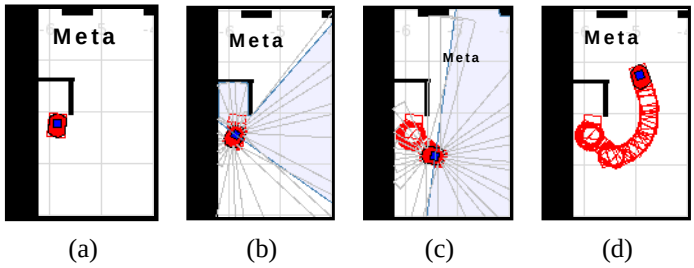


Figura 6.12: Robot identificando una trampa y saliendo de élla. (a) Posición inicial, (b),(c),(d) el robot sale de la trampa (e) el robot se orienta y va hacia la meta

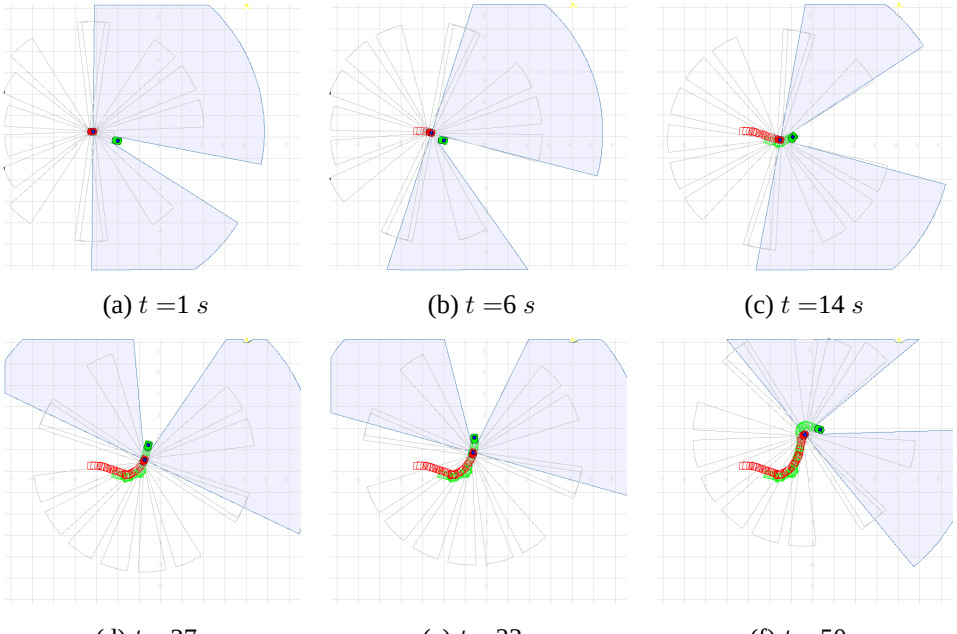
Una vez que encuentra una zona libre frente a él, el robot deambula (Figura 6.12 (c))

y encuentra una zona de orientación. El robot ahora puede orientarse y llegar a la meta (Figura 6.12 (d)). El experimento tiene éxito si el robot puede reconocer la trampa y girar hasta que ya no tiene la pared frente a él. La estrategia para salir de la trampa no es infalible. Al igual que el PTR para orientarse, si el espacio es reducido, tendrá complicaciones y es posible que no salga.

6.2.4 *Seguir* a un objeto móvil

En esta prueba el robot tiene que seguir a otro robot móvil al menos por 10 m. El mapa de prueba es un área libre como el mostrado en la Figura 6.13. El robot a ser seguido o "líder" es controlado por el usuario con una palanca de control (*joystick*). Si el robot seguidor no sabe qué acción tomar entonces el experimento falla. Se realizaron 40 experimentos y se logró una precisión de 97.5 %.

Las Figuras 6.13 (a)-(i) muestran a un robot que es seguido por otro robot. El PTR busca un objeto a seguir (*i.e.*, una marca cercana) dentro de la zona de seguimiento. Cuando el robot percibe el objeto, se aproxima hacia él hasta llegar a la distancia mínima permitida (0.3 m) y se detiene. Si el objeto se aproxima al robot, éste retrocede para evitar colisiones. El robot puede identificar si el objeto a seguir se mueve a la derecha o a la izquierda y reacciona acorde a este movimiento.



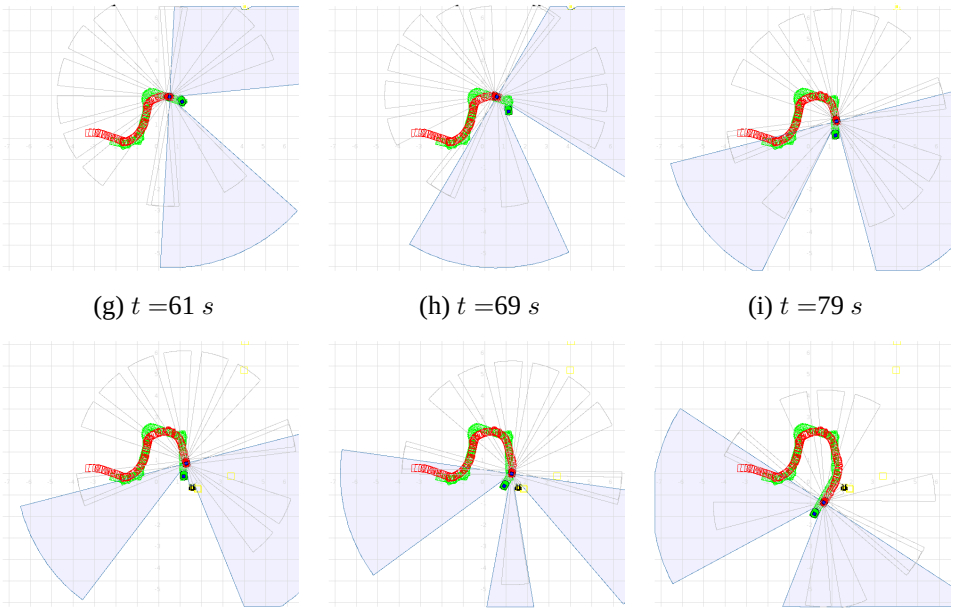
(a) t =1 s (b) t =6 s (c) t =14 s

(d) t =27 s (e) t =33 s (f) t =50 s

Figura 6.13: ***Seguir*** a otro robot móvil. Parte 1.

En las Figuras 6.14 (j)-(l) se muestra que cuando un objeto aparece, el robot líder `gira a la derecha` para evadirlo y el robot seguidor hace lo mismo.

Cuando el objeto está lejos pero dentro del alcance de la zona de seguimiento, el robot se desplaza a una mayor velocidad. Sin embargo, si el objeto está cerca del robot, éste se desplazar a menor velocidad. Si el objeto se aleja de la zona de seguimiento, el robot seguidor no puede buscarlo pues carece de la habilidad para éllo.



(g) t =61 s (h) t =69 s (i) t =79 s

(j) t =86 s (k) t =99 s (l) t =117 s

Figura 6.14: ***Seguir*** a otro robot móvil. Parte 2.

El robot reacciona acorde a los movimientos del objeto que se encuentra en la zona de seguimiento. Sin embargo, no cuenta con la habilidad de evasión de obstáculos por lo que mientras sigue al objeto que está en la zona, puede tener colisiones. Las velocidades de ambos robots deben ser similares para que el robot seguidor no pierda constantemente al objeto que va siguiendo.

La Tabla 6.1 resume los resultados en simulación de los PTRs básicos. Se muestra el número de tareas para probar cada PTR y la precisión.

Una vez probados los PTRs básicos, se realizaron experimentos con los PTRs que combinan dos o más de éllos. A continuación se describen las pruebas.

PTR	Pruebas	Precisión
Deambular	30	86.67 %
Orientarse	30	100.00 %
Salir de trampa	30	80.00 %
Seguir	40	97.50 %

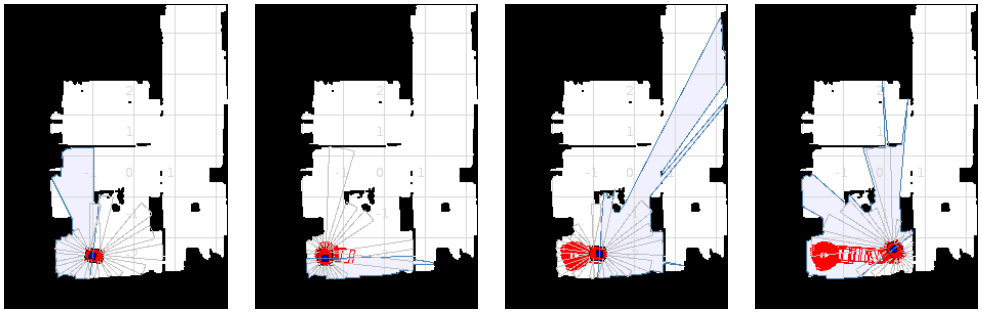
Tabla 6.1: Precisión: PTRs básicos en simulación

6.2.5 *Ir a un punto* (*orientar* + *deambular*)

El PTR para ***ir a un punto*** utiliza los dos PTRs básicos ***deambular*** y ***orientarse*** aprendidos previamente. Para lograr su meta, este PTR compuesto es capaz de manejar las metas para cada PTR básico de acuerdo al estado percibido. Si el robot se está moviendo hacia su destino y tiene que alejarse de un obstáculo, el robot puede volver a orientarse nuevamente hacia la meta. Se realizaron 30 pruebas, 3 por cada mapa. En 20 de los experimentos existieron objetos dinámicos en el ambiente, se obtuvo una precisión del 86.67 %. A continuación se describen tres casos: 1) el robot tiene que salir de una habitación teniendo como punto final el extremo de un pasillo, 2) el robot partiendo del extremo de un pasillo y como punto final una habitación y 3) el robot tiene que atravesar varias habitaciones para llegar a su punto final.

Caso 1: Saliendo de una habitación

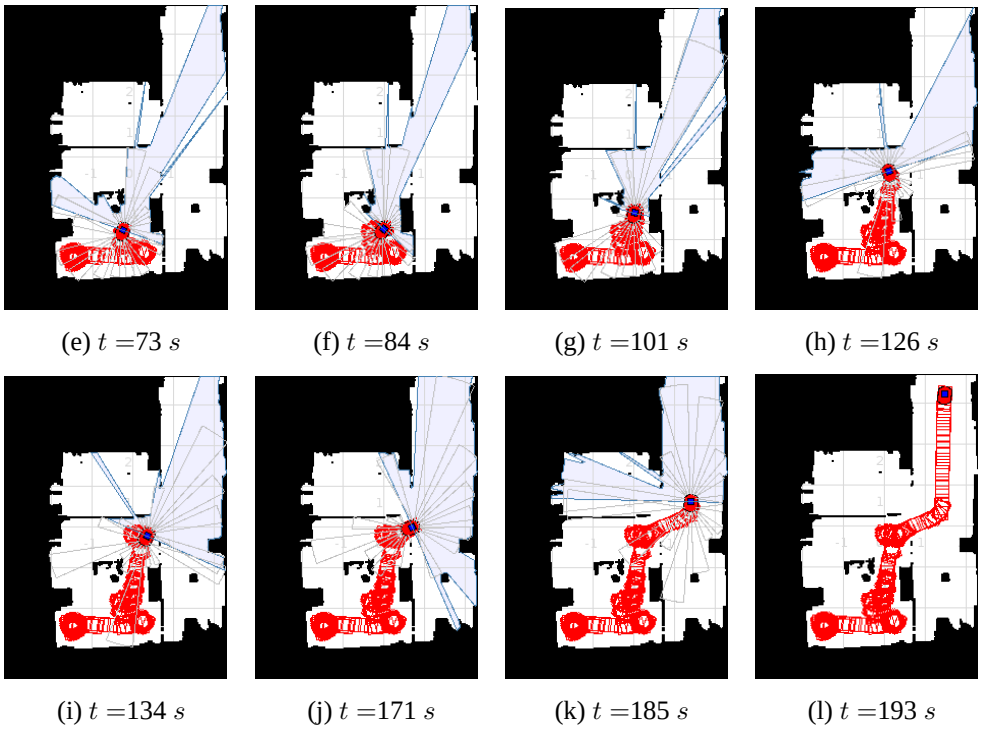
En esta prueba el robot tiene que salir de una habitación hasta el extremo de un pasillo. El robot se encuentra inicialmente en dirección opuesta a su punto final (Figura 6.15 (a)). Las paredes se encuentran muy cerca al robot por lo que el robot `deambula` y sale de la esquina (Figuras 6.15 (a)-(c)) hasta encontrarse con una pared.



(a) t =1 s (b) t =14 s (c) t =37 s (d) t =60 s

Figura 6.15: ***Ir a un punto***: saliendo de una habitación. Parte 1.

El robot `gira a la izquierda` (Figura 6.15 (d)) y ***deambula*** hasta encontrar una zona de orientación (Figura 6.16 (e)) en la cual se puede orientar hacia la meta. El robot se desplaza por un espacio estrecho limitado por un mueble y la pared (Figuras 6.16 (f)-(h)). Al llegar a la puerta, el robot vuelve a ***orientarse*** (Figura 6.16 (i)) y sale al pasillo (Figura 6.16 (j)). Encontrándose en el pasillo, el robot se `orienta` hacia la meta y `avanza` hasta llegar al extremo (Figuras 6.16 (k)-(l)).



(e) t =73 s (f) t =84 s (g) t =101 s (h) t =126 s

(i) t =134 s (j) t =171 s (k) t =185 s (l) t =193 s

Figura 6.16: ***Ir a un punto***: saliendo de una habitación. Parte 2.

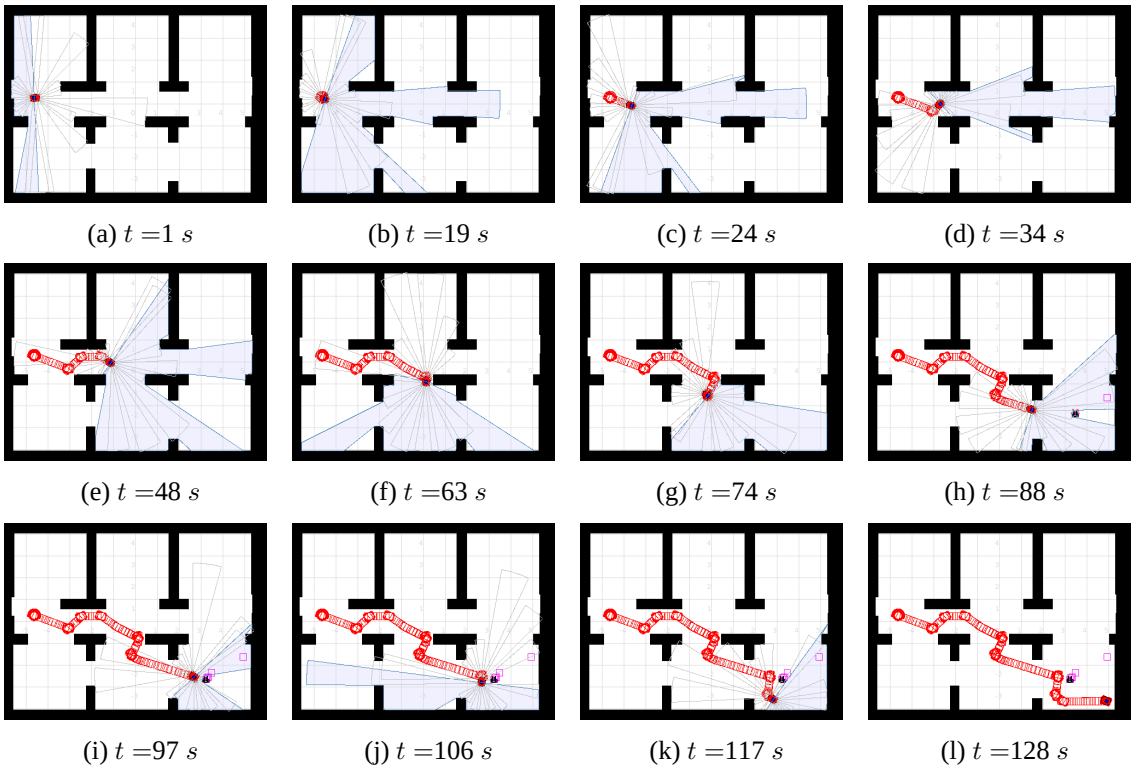
El robot logró salir de una habitación pasando por espacios estrechos originados por muebles ubicados en el centro del cuarto. Una vez en zonas más despejadas, el robot se orientó, pudo salir de la habitación y avanzar por el pasillo hasta llegar a su punto final. En espacios estrechos el robot realiza giros innecesarios, su comportamiento es inestable aunque logra salir.

Caso 2: De un pasillo a una habitación

En este caso, el robot tiene como origen el extremo de un pasillo y su destino final es una habitación pero a diferencia del caso anterior en el cual existía una única entrada, ahora

tiene que pasar frente a varias.

El robot inicia orientado en dirección opuesta a su punto meta (Figura 6.17 (a)) y `gira a la izquierda` para ***orientarse*** (Figura 6.17 (b)). El robot encuentra espacio libre al frente y `avanza` (Figura 6.17 (c)) pero se encuentra con una esquina a una distancia cercana y `gira a la izquierda` para evadirla (Figura 6.17 (d)). El robot deambula y entra a una habitación donde vuelve a orientarse hacia la meta (Figuras 6.17 (e)-(h)). El robot entra a la habitación destino pero aparece un objeto al cual evade (Figuras 6.17 (i)-(k))girando a la derecha y logra llegar a la esquina del cuarto que es su punto meta (Figura 6.17 (l)).



(a) t =1 s (b) t =19 s (c) t =24 s (d) t =34 s

(e) t =48 s (f) t =63 s (g) t =74 s (h) t =88 s

(i) t =97 s (j) t =106 s (k) t =117 s (l) t =128 s

Figura 6.17: ***Ir a un punto***: de un pasillo a una habitación

El robot llegó a su meta de forma satisfactoria. No hubo colisiones y el robot logró evadir un obstáculo móvil que se le presentó al final del trayecto. El avance del robot por el pasillo no fué estable porque el robot percibió las esquinas como obstáculos y los giros que realizó para evadirlas colocó al robot en posiciones que generaron un desplazamiento irregular.

El robot presenta un desplazamiento seguro en el ambiente y es capaz de evadir obstáculos inesperados sin perder su objetivo final que es llegar a un punto meta. La desventaja de este enfoque es que el robot resuelve lo que se le va presentando sin considerar metas posteriores.

De esta manera, es posible que al realizar una evasión, se desvíe de tal modo que tome caminos más largos para llegar a la meta. De forma similar a los experimentos con los PTRs básicos, en ambientes abiertos el robot muestra un comportamiento más estable que en ambientes reducidos. Este PTR no incluye la habilidad de ***salir*** de trampa por lo que en ocasiones, el robot entraba a espacios estrechos de los cuales ya no podía salir.

Se realizaron experimentos combinando diversos PTRs básicos. Los PTRs resultantes son variantes del PTR ***ir a un punto*** y ***seguir***:

- ***Deambular*** + ***orientarse*** + ***salir de trampa***. Se realizaron 30 experimentos, 3 por cada mapa de prueba y se obtuvo una precisión de 93.33 %. El desempeño del robot mejoró pues logró llegar a metas a las que no podía llegar por quedar atrapado en espacios estrechos.

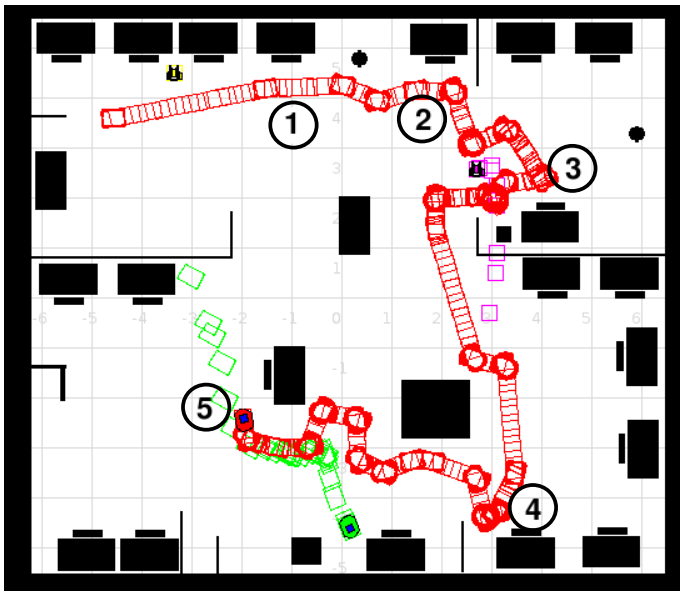
- ***Seguir*** + ***deambular***. El objetivo de esta prueba es que el robot siga a un objeto y evada obstáculos. Se realizaron 40 experimentos, 20 en el mapa mostrado en la Figura 6.3(a) y 20 en el mapa mostrado en la Figura 6.3(b) pues son los que cuentan con más espacio abierto. Se obtuvo una precisión del 90 %.

- ***Seguir*** + ***deambular*** + ***salir de trampa***. Similar al anterior pero con la habilidad de salir de trampas al deambular. Se efectuaron 40 experimentos en los mismos mapas del punto anterior y se obtuvo el 100 % de precisión.

6.2.6 Visitar distintos lugares

En esta tarea se utilizó el PTR ***ir a un punto*** para ir a distintos puntos en un ambiente. En su aplicación original, este PTR recibe inicialmente un punto origen y un punto meta. Para visitar distintos puntos, el estado enviado al robot es modificado para que al llegar a cada sub-meta ésta se convierta en estado inicial del siguiente tramo. Este proceso se realiza hasta llegar al último punto de la lista. Una de las ventajas de los PTRs es su flexibilidad pues no fue necesario modificar el PTR para que ejecutara la variante de la tarea. Se modificó el proceso externo que genera el estado para efectuar el cambio de las metas una vez cumplidas.

La Figura 6.18 muestra un ejemplo de esta tarea. El robot tiene que ir a los cinco puntos mostrados. En el trayecto del punto 2 al punto 3 tiene que evadir un obstáculo que le apareció de improviso. Del punto 3 al 4 el robot se orienta hacia el punto 4 pero en el camino encuentra un obstáculo fijo cuadrado al cual tiene que rodear. En su recorrido del punto 4 al punto 5 se le atraviesa un robot, lo evade y finalmente llega al punto final. Se realizaron 30 pruebas, 3 por cada mapa de prueba, obteniéndose una precisión del 96.67 %.



Visitando 5 lugares

Figura 6.18: (a) En (2-3), el robot se aleja de un obstáculo. En (4-5) el robot evade a otro robot.

La Tabla 6.2 resume los resultados en simulación de las pruebas con PTRs jerárquicos. Se muestra el número de tareas y la precisión.

PTR	Pruebas	Precisión
deambular + ***orientarse*** (ir a un punto)	30	86.67 %
deambular + ***orientarse*** + ***salir de trampa***	30	93.33 %
seguir + ***deambular***	40	90.00 %
seguir + ***deambular*** + ***salir de trampa***	40	100.00 %
visitar 5 lugares	30	96.67 %

Tabla 6.2: Precisión de losPTRs jerárquicos en simulación

6.3 Markovito: el robot de servicio

El siguiente paso consiste en probar los PTRs en un robot real. El robot usado para este fin es Markovito, un robot móvil de servicio. Para Markovito se desarrolló una plataforma general y flexible basada en distintos módulos de propósito general para ejecutar tareas, tales como planeación, navegación, localización, localización de objetos, interacción por voz, etc. Estos módulos están diseñados para que puedan ser adaptados fácilmente y combinados para

(a) Laboratorio de robótica del INAOE

(b) piso 2 del edificio 8 del INAOE

(c) UPAEP, Concurso Mexicano de Robótica 2007

(d) Palacio de Minería, Torneo Mexicano de Robótica 2008

Figura 6.19: Mapas de ambientes reales

realizar distintas tareas. El módulo para navegación de Markovito son los PTRs aprendidos en esta tesis. Los módulos son coordinados por un controlador basado en procesos de decisión de Markov (MDP). Esta plataforma ha sido desarrollada por diversos estudiantes de maestría y doctorado. Información detallada sobre la arquitectura, módulos y el coordinador, puede consultarse en [Avilés et al., 2009] y [Avilés et al., 2007].

Markovito es un robot PeopleBot de ActivMedia [Robotics, 2006] que cuenta con los siguientes sensores: (i) sensor LASER SICK LMS200, (ii) anillo de 16 sonares en la parte inferior, (iii) sistema de visión estéreo STH-HMCS2-VAR-C.

En Markovito se probaron dos PTRs, el básico para ***seguir*** a un objeto móvil y el jerárquico ***ir a un punto***. Las pruebas se realizaron en: (i) el piso 2 del edificio 8 del INAOE, (ii) el laboratorio de robótica del INAOE, (iii) en instalaciones de la UPAEP durante el Concurso Mexicano de Robótica (2007) y (iv) en el Palacio de Minería durante el Torneo Mexicano de Robótica (2008). La Figura 6.19 muestra los mapas de los ambientes reales de prueba.

6.3.1 PTRs como módulo de navegación

Los PTRs aprendidos se integraron como el módulo de navegación en Markovito. Las tareas dadas a Markovito son:

- Navegar a varios lugares designados semánticamente[TMR08nab-url, 2007]. Al mapa del ambiente se le asocian etiquetas como: cocina, librero, vitrina. Al robot se le proporciona la lista de lugares a visitar para que utilizando el PTR para ***ir a un punto*** realice la tarea. En la Figura 6.20 se muestra uno de los ambientes reales en los que

se probó el robot. En esta tarea, Markovito tiene que avanzar de forma segura por tres lugares definidos al inicio de la prueba y regresar a su posición de origen. Las Figuras 6.21 (a) y (b) muestran a Markovito navegando por el ambiente y alcanzando los puntos destino. El ambiente es distinto a los de entrenamiento. La estrategia de navegación consiste en seguir un camino en el que los puntos son las sub-metas y los lugares a visitar son los puntos principales a alcanzar.

(a) Ambiente real (b) Mapa etiquetado

Figura 6.20: El mapa (a) corresponde al ambiente real para la prueba. Se agregó texto a la imagen (nombres de los lugares) para mayor claridad. El mapa (b) es el que se obtuvo y se etiquetó para la prueba

- Seguir a una persona. La prueba consiste en seguir a una persona. Para el control del robot se adaptó el PTR para seguimiento de un objeto móvil. En la sección 6.3.2 se describe la adaptación.

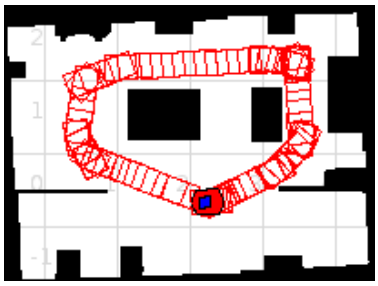
- Encontrar un objeto de entre un conjunto de distintos objetos. En esta tarea, a Markovito se le muestra el objeto que posteriormente tiene que encontrar. Markovito tiene que navegar al área indicada y buscar el objeto. El robot envía un mensaje al reconocer el objeto. Para esta tarea, la estrategia de navegación consiste en seguir un camino cuyos puntos se colocan en zonas estratégicas. Cada vez que Markovito alcanza un punto se detiene, busca el objeto y navega al siguiente punto. La Figura 6.21(c) muestra a Markovito buscando el objeto [TMR07buscar-url, 2007].

- Entregar mensajes y/o objetos entre distintas personas. La meta en esta tarea es recibir y entregar un mensaje, un objeto o ambos a petición del usuario. La Figura 6.21(d)

muestra a Markovito entregando una cerveza.

(a) Visitando distintos lugares (b) Visitando distintos lugares

(c) Buscando un objeto (d) Entregando una cerveza



(e) Trayectoria seguida en (a) y (b)

Figura 6.21: Markovito ejecutando diversas tareas de navegación

Las primeras tres tareas son parte del concurso RoboCup@Home [Robocup url, 2008]. La Tabla 6.3 muestra el número de pruebas realizadas, el número de intervenciones del operador y la precisión. De las 120 pruebas realizadas, 50 se ejecutaron en el laboratorio de robótica del INAOE (mapa 6.19(a)); 60 pruebas se efectuaron en el piso 2 del edificio 8 del INAOE (mapa 6.19(b)); 5 pruebas se realizaron en la UPAEP (mapa 6.19(c)) y 5 en el Palacio de Minería (mapa 6.19(d)). Los ambientes son áreas divididas por mamparas simulando habitaciones. Se utilizó mobiliario de oficina: sillas, escritorios, mesas, macetas y cajas principalmente. Las intervenciones del operador consistieron básicamente en quitar

objetos fuera del alcance del sensor laser y sacar al robot de lugares estrechos. El robot mostró consistencia en su comportamiento en los 4 ambientes.

	Pruebas	Intervenciones	Precisión
Ir a un punto	120	20	85.00 %

Tabla 6.3: Markovito: Precisión para ir a un punto

6.3.2 Seguir a una persona

El PTR aprendido originalmente para seguir a un objeto móvil fue hecho para un sensor láser y para seguir a otro robot móvil. Sin embargo, el seguimiento de una persona involucra diversas dificultades, por ejemplo, al caminar se forma un espacio en el cual el sensor láser por un momento no detecta nada, originando un comportamiento inestable, es decir intercalando acciones como `avanzar`, `girar-derecha`, `girar-izquierda`, `avanzar`. Para lograr un comportamiento robusto en el cual el robot se desplace sin vacilación, sin oscilar entre acciones, se reemplazó el proceso de identificación de marcas (*i.e.*, paredes, esquinas y discontinuidades) por un proceso de identificación de marcas visuales [Sánchez-Texis, 2008].

El enfoque relacional utilizado para aprender los PTRs facilitó el reemplazo ya que el único requisito era que el sensor, en este caso una cámara, devolviera los atributos de orientación y distancia de la marca a seguir. Para realizar la tarea se probó de dos formas: (i) utilizando un cartel que la persona mostraba al robot al caminar y (ii) utilizando la ropa de la persona para un seguimiento más natural que con la persona llevando un cartel. El comportamiento del robot es muy similar al observado en el seguimiento con laser como puede verse en la Figura 6.22. En este ejemplo, se le muestra un cartel al robot para que el proceso visual extraiga la marca a seguir. La persona guía al robot por entradas, áreas abiertas, lo hace girar en diversas direcciones y regresa al lugar de origen. Cuando el robot no percibe a la persona en la zona de seguimiento, emite un mensaje: “no puedo verte” para que la persona se coloque en una posición adecuada (ver video [TMR08seguir-url, 2008]).

En un ambiente real existen factores no contemplados en simulación que pueden afectar el desempeño del robot. Por ejemplo, los manteles que cubren el mobiliario puede generar un comportamiento inestable en el robot pues al ser agitados por el aire, el robot percibe de forma intermitentemente un obstáculo. Otro factor que puede influir es la textura del piso. Si no es liso, el robot puede calcular que llegó a la meta antes de alcanzarla pues su odometría se ve afectada. La Tabla 6.4 resume los resultados en el robot real. Se muestra el número

Figura 6.22: Seguir a una persona. El PTR de seguimiento se adaptó para seguir a una persona. Ésto se hizo reemplazando el proceso de identificación de marcas utilizando un sensor láser por un proceso de identificación de marcas visuales. La persona muestra una imagen al robot para que el proceso extraiga la marca que debe seguir. La persona guía al robot por distintas partes del ambiente y regresa a su lugar de origen.

de pruebas realizadas para cada PTR, la precisión lograda y el número de intervenciones del operador.

	Pruebas	Intervenciones	Precisión
Seguir a una persona	20	3	90.00 %

Tabla 6.4: Markovito: Precisión para seguir

6.4 Comparación: Reactivo / Deliberativo

La navegación con PTRs no tiene por objetivo llevar al robot por la ruta óptima. Es un enfoque totalmente reactivo en el cual el robot va comportándose de acuerdo a lo que se le va presentanto en el ambiente. Una de las principales ventajas de este enfoque es que en ambientes desconocidos y dinámicos el robot es capaz de navegar sin chocar con obstáculos independientemente de su forma y tamaño, reconocer y salir de trampas, sin perder la meta. Sin embargo, la misma naturaleza reactiva origina que el robot pueda caer en situaciones en

las que debe ejecutar más acciones de las necesarias para lograr su objetivo. En esta sección se realiza una comparación entre el enfoque totalmente reactivo que se utiliza en la tesis y un método deliberativo.

6.4.1 Iteración de Valor

Determinar la trayectoria a seguir para llegar a un punto meta puede verse como un problema de decisión secuencial puesto que el robot debe saber qué secuencia de acciones lo llevará al destino. La especificación de un problema de decisión secuencial para un ambiente completamente observable utilizando un modelo de transiciones y recompensas es llamado Proceso de Decisión de Markov o MDP [Sutton y Barto, 1998].

Supongamos que nuestro mapa del ambiente está representado por celdas, donde cada celda es un posible estado. El robot inicia en una celda y en cada una, el robot puede realizar un conjunto de acciones (*e.g.*, `izquierda`, `derecha`, `arriba`, `abajo`). El ambiente considerado no es determinístico por lo que se especifican probabilidades de transición de un estado a otro para cada acción en cada posible estado. A esta especificación se le llama modelo de transición. Por ejemplo, la acción se realiza correctamente con 0.8 de probabilidad, pero con 0.2 de probabilidad el robot se moverá de otra manera. La definición del problema requiere también una función de utilidad que especifique una recompensa que puede ser positiva o negativa, para cada estado. La utilidad de una secuencia de estados es la suma de todas las recompensas sobre la secuencia.

La solución de un MDP debe especificar qué es lo que el robot debe hacer para cualquier estado en el que se encuentre. A esta solución se le llama política, generalmente denotada por π; $\pi(s)$ y es la acción recomendada por la política π para un estado s. Una política óptima ($\pi*$) es aquella que tiene la máxima utilidad de la secuencia de estados encontrada al ser ejecutada. La utilidad de un estado es la utilidad esperada del estado de secuencias encontrado cuando una política óptima es ejecutada, iniciando en ése estado.

Un algoritmo para encontrar políticas óptimas es iteración de valor [Sutton y Barto, 1998]. La idea básica del algoritmo es calcular la utilidad de cada estado y usar esas utilidades para seleccionar una acción óptima para cada estado. Para utilizar este algoritmo en navegación, el mapa del ambiente se convirtió a un mapa de celdas, donde cada una mide 0.05 m $\times$ 0.05 m. El algoritmo calcula las rutas óptimas desde cada una de las celdas hasta la celda meta, de esta manera, al darle el punto origen el robot conoce previamente los puntos que va a recorrer. Se realizaron experimentos con ambos enfoques, el reactivo (PTRs) y el deliberativo al cual nos referiremos como DELIB. La siguiente sección muestra los resultados, así como

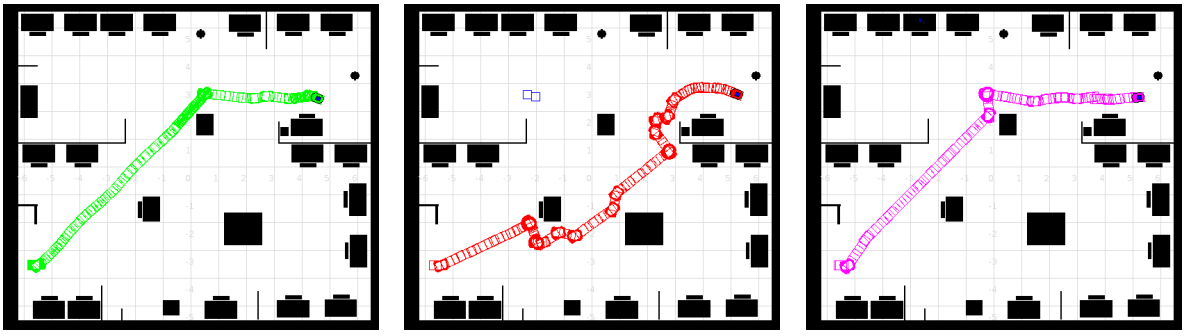
ventajas y desventajas de cada uno.

6.4.2 Experimentos

Se hicieron tres tipos de pruebas de navegación con el fin de ver las ventajas y desventajas de cada enfoque: (i) con la política óptima obtenida con iteración de valor (DELIB), (ii) usando PTRs y (iii) combinando una política óptima con PTRs. Un robot está controlado por PTRs, otro robot sigue una trayectoria dada por el algoritmo de iteración de valor (DELIB) y un tercer robot genera la trayectoria con DELIB y el desplazamiento es controlado por PTRs (PTRs + DELIB). A continuación se describen tres casos: 1) ventaja del enfoque deliberativo, 2) ventaja del enfoque reactivo, y 3) trayectorias similares.

Caso 1: ventaja del enfoque deliberativo (DELIB)

En esta prueba, el robot debe llegar a un punto meta en un ambiente donde sólo hay obstáculos fijos. En la Figura 6.23 (a) se muestra que robot que usa DELIB genera una trayectoria del punto inicial al punto final y la sigue. No tiene problemas con los obstáculos pues no hay cambios en el ambiente. Por otro lado, la Figura 6.23 (b) muestra cómo el robot que usa PTRs se desplaza de acuerdo a lo que percibe en el ambiente. El robot tiene que evadir un obstáculo de forma cuadrada al inicio de su trayecto, orientarse y avanzar rumbo al punto meta y evadir una pared. El robot que usa PTRs tuvo que girar en diversos puntos de su camino, mientras que el robot que usa DELIB giró sólo en 1 punto. Un mayor número de giros puede generar que el robot se desvíe más fácilmente y tenga que volverse a orientar.



(a) DELIB (b) PTR (c) DELIB+PTR

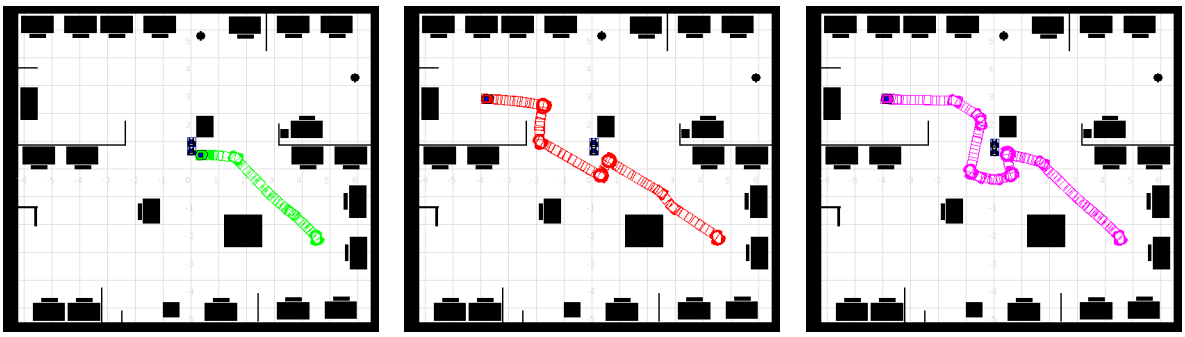
Figura 6.23: Ventaja del enfoque deliberativo. En (a) el robot genera una trayectoria en la que el robot efectúa menos giros, lo que favorece un desplazamiento más estable y preciso para alcanzar la meta. Las trayectorias generadas por éste enfoque son además, las más cortas

El robot que usa el enfoque combinado genera la trayectoria usando DELIB y luego sigue los puntos usando PTRs. La Figura 6.23 (c) muestra que el robot no se encontró obstáculos en el camino. A diferencia de la trayectoria seguida por DELIB la trayectoria del robot con enfoque combinado tiene diferencias porque al seguir los puntos o centros de cada celda usando PTRs el robot se posiciona diferente para alcanzarlos.

El robot que utiliza el enfoque deliberativo genera una trayectoria en la que el robot efectúa menos giros, lo que favorece un desplazamiento más estable y preciso para alcanzar la meta. Las trayectorias generadas por éste enfoque son además, las más cortas.

Caso 2: ventaja del enfoque reactivo (PTRs)

En esta prueba, el robot debe llegar a un punto meta en un ambiente donde se presenta un obstáculo que no existía cuando el robot que usa DELIB generó su trayectoria. En la Figura 6.24 (a) se muestra que el robot se detiene pues el punto al que se dirigía ya no es libre y no cuenta con otra alternativa. El robot llega a la meta. Por otro lado, en la Figura 6.24 (b) se muestra que el robot que usa PTRs logra llegar a la meta. El robot con enfoque combinado también llega a la meta aunque las trayectorias tienen diferencias. El robot con enfoque combinado evade únicamente un obstáculo mientras que el robot que usa PTRs tiene que evadir dos obstáculos, originando desviaciones con respecto a la trayectoria óptima.



(a) DELIB (b) PTR (c) DELIB+PTR

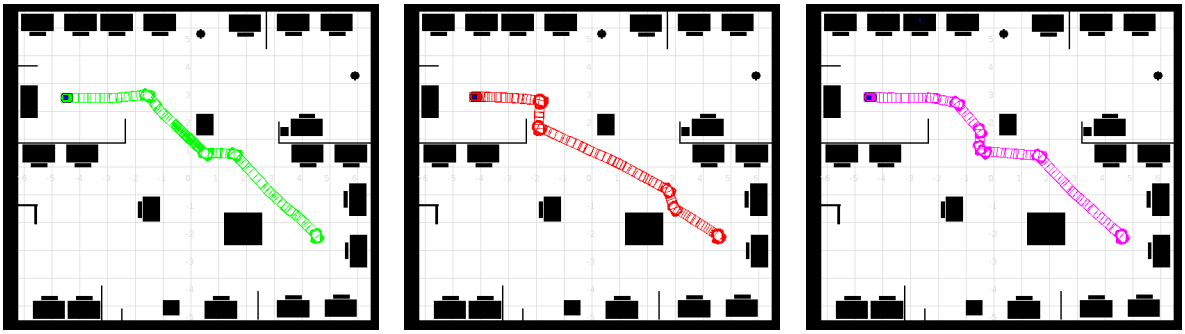
Figura 6.24: Ventaja del enfoque reactivo. En (b) el robot que tiene la capacidad de reaccionar ante obstáculos inesperados y logra llegar a la meta aunque no lo haga con una trayectoria óptima, pues no es esto su objetivo

El robot que usa PTRs tiene la capacidad de reaccionar ante obstáculos inesperados y logra llegar a la meta aunque no lo haga con una trayectoria óptima, pues no es esto su

objetivo. Se observa que una combinación de los enfoques puede ser útil pues el robot puede reaccionar a cambios en el ambiente y se desvía menos de una trayectoria óptima.

Caso 3: trayectorias similares

En este caso se muestra que no necesariamente un enfoque es el mejor para todas las situaciones. En las Figuras 6.25 (a), (b) y (c) se observa que los tres robots llegan a la meta utilizando trayectorias similares. Las diferencias propias de cada enfoque son las originan los cambios en las trayectorias.



(a) DELIB (b) PTR (c) DELIB+PTR

Figura 6.25: Trayectorias similares. En (a), (b) y (c) los robots logran llegar a la meta siguiendo trayectorias similares

Se efectuaron 30 experimentos para cada enfoque, 3 por cada mapa de prueba. De los 30 experimentos en 15 se agregaron objetos dinámicos y 15 se realizaron en ambientes estáticos. El experimento se considera exitoso si el robot puede llegar a su meta. Los PTRs lograron una precisión de 83.33 %; usando el algoritmo de iteración de valor se alcanzó una precisión de 50.00 % ya que en las pruebas con obstáculos inesperados falló; y al utilizar la combinación de ambos se mejoró la precisión a 90.00 %. La Tabla 6.5 muestra una comparación entre los distintos enfoques. La tabla muestra si el enfoque es capaz de manejar obstáculos dinámicos, así como la planeación de una ruta óptima del punto origen al punto destino. Se muestra también la distancia promedio recorrida en cada prueba y el porcentaje recorrido con respecto a la ruta óptima. Usando PTRs se recorre un porcentaje mayor de distancia que con los otros enfoques. Con DELIB la distancia recorrida es la óptima, sin embargo cuando existen obstáculos inesperados no logra llegar por eso el porcentaje es menor al 100.00 %. Cuando se combinan los PTRs con DELIB el recorrido es menor que sólo con PTRs pero a diferencia del DELIB, el robot logró llegar a la meta con mayor precisión.

	PTR	DELIB	DELIB+PTR
Obstáculos dinámicos	Si	No	Si
Ruta óptima	No	Si	Si
Tareas	30	30	30
Precisión	83.33 %	50.00 %	90.00 %
Distancia promedio recorrida (m)	17.4	12.07	14
Con respecto a la ruta óptima(%)	144.16 %	100 %	115.99 %

Tabla 6.5: Comparación: PTRs, DELIB y PTRs+DELIB

Las características de la Tabla 6.5 pueden representarse de forma cualitativa para visualizar con más claridad los enfoques. La Tabla 6.6 muestra las características consideradas y la discretización de sus valores. La Tabla 6.7 muestra los valores asignados a cada característica. Con estos valores se construyeron las gráficas mostradas en la Figura 6.26.

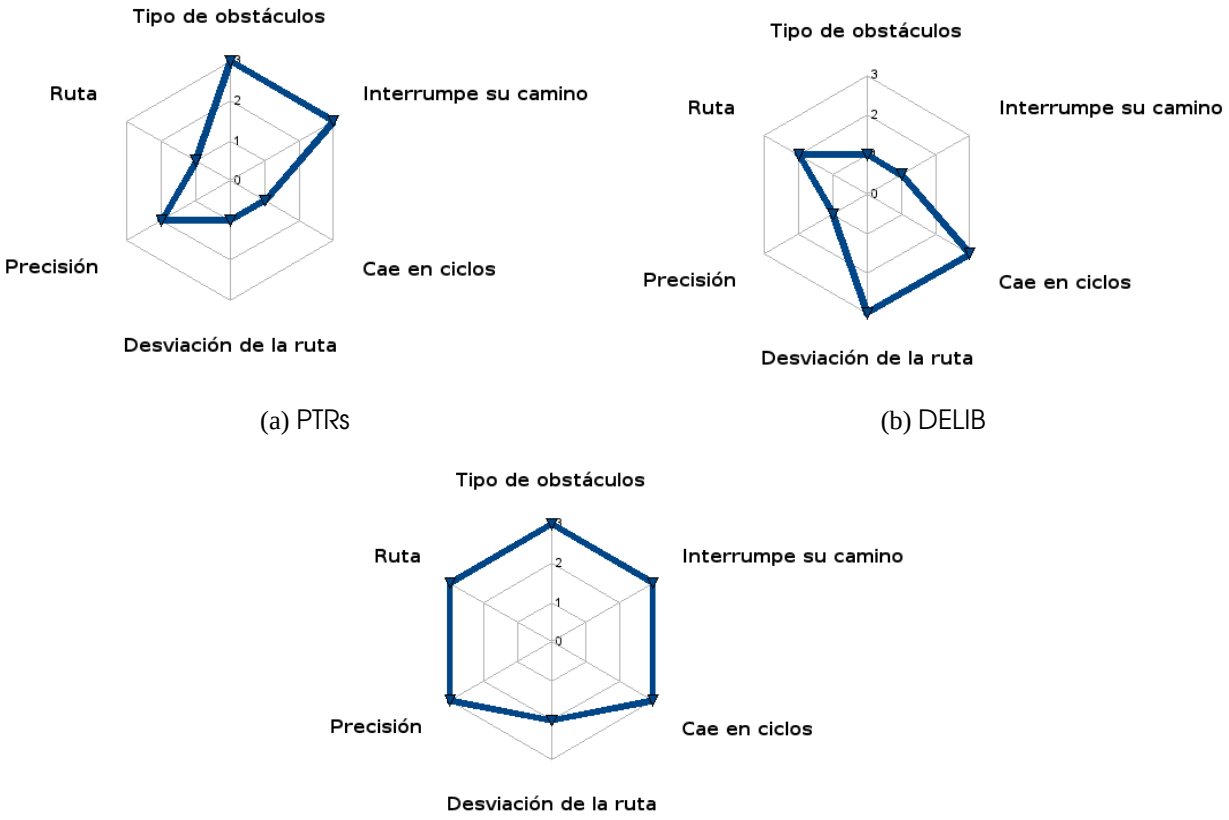
	Valores		
Características	1	2	3
Tipo de obstáculo	Ninguno	Estáticos	Dinámicos
Generación de ruta	No-planeada	Planeada	Ambas
Precisión	50 %-0 %	94 %-49 %	100 %-95 %
Desviaciones de la ruta óptima	mayor a 20 %	1 %-20 %	0 %
Cae en ciclos	Si	–	No
Interrumpe su camino	Si	–	No

Tabla 6.6: Características y rango valores: PTRs, DELIB y PTRs+DELIB

	DELIB	PTR	DELIB+PTR
Tipo de obstáculo	1	3	3
Generación de ruta	2	1	3
Precisión	1	2	3
Desviaciones de la ruta óptima	1	2	3
Cae en ciclos	3	1	3
Interrumpe su camino	3	3	3

Tabla 6.7: Asignación de valores: PTRs, DELIB y PTRs+DELIB

Los PTRs dan al robot la ventaja de reaccionar ante eventos inesperados, en ambientes desconocidos. Cuando el robot usa DELIB el algoritmo genera la política para un ambiente conocido por lo que al presentarse un obstáculo en una celda que previamente estaba libre,



(a) PTRs

(b) DELIB

(c) PTRs + DELIB

Figura 6.26: Comparación cualitativa. Las gráficas (a) y (b) muestran que los enfoques reactivo (PTRs) y deliberativo DELIB poseen características que son una ventaja o desventaja dependiendo de las situaciones y el objetivo a alcanzar. Una combinación de ambos (c) aparentemente mejora el comportamiento del robot.

la política ya no es la correcta. Para usar DELIB se requiere generar nuevamente la política óptima y conocer completamente el ambiente. En contraste, los PTRs no requieren conocer el mapa completo para poder llegar a la meta.

Si en el mismo ambiente al robot se le da otro punto inicial y otra meta, el robot con delib tiene que volver a generar la trayectoria mientras que el que usa PTRs no.

Una desventaja de los PTRs es que pueden conducir al robot a situaciones en las que se desvíe y recorra una trayectoria más larga antes de encontrar una zona para orientarse que le permita retomar su camino a la meta, inclusive puede recorrer la misma ruta de forma cíclica y no salir de élla. Por el contario, un robot con DELIB recorre la trayectoria óptima y no realiza giros innecesarios.

El enfoque combinado muestra resultados más robustos particularmente en trayectos donde los espacios son reducidos o saturados de objetos. Sin embargo, cuando los espacios son abiertos y se le presenta un obstáculo, puede generar trayectorias más complicadas. Una alternativa a este comportamiento es volver a generar una trayectoria óptima a partir de la celda libre después de que evadió el obstáculo.

6.5 Clasificación de ademanes

En el capítulo 5 se describió cómo aprender gramáticas. Se describió la representación de las secuencias de ademanes y se indujeron nueve gramáticas, una por cada uno de los nueve ademanes: `atención`, `acercar`, `izquierda`, `derecha`, `detenerse`, `girar-hacia-la-derecha`, `girar-hacia-la-izquierda`, `saludar` y `apuntar`. En esta sección se prueban las gramáticas aprendidas para clasificación, usando los conjuntos de prueba que se obtuvieron. Se recuerda el proceso que se siguió para la construcción de los conjuntos de entrenamiento y prueba:

- De 50 secuencias por ademán, se seleccionaron aleatoriamente 20 secuencias para inducir las gramáticas. Las 30 secuencias restantes constituyeron el conjunto de prueba.
- Del conjunto de prueba se generaron sub-conjuntos: seleccionar aleatoriamente (i) 2 secuencias y (ii) 10 secuencias. El objetivo es comparar la precisión de las gramáticas aprendidas al incrementar el número de secuencias de entrenamiento.
- Se seleccionó aleatoriamente un conjunto de ejemplos negativos para aprender las gramáticas. Para cada ademán, los ejemplos negativos son secuencias de los ademanes restantes. El conjunto de ejemplos negativos se construyó tomando un ejemplo de cada ademán restante.
- De cada sub-conjunto se aprendió una gramática para cada ademán, obteniendo así 9 gramáticas.
- Se mostró cada secuencia del conjunto de pruebas a las gramáticas. La gramática que obtiene una evaluación mayor corresponde al ademán mostrado.
- Este proceso se repitió 10 veces para cada sub-conjunto.

El número o proporción de ejemplos identificados para cada clase se registraron en una matriz de confusión. De este modo, se puede observar en qué clases se concentran los errores de clasificación. A partir de la matriz de confusión se calculó el porcentaje de reconocimiento o precisión de las gramáticas inducidas. La precisión se define según se expresa en la ecuación 6.2.

$$\text{precisión} = \frac{\text{no. de ademanes clasificados correctamente}}{\text{no. de ademanes de prueba}} \times 100 \tag{6.2}$$

La Tabla 6.8 muestra la precisión global obtenida por FOSeq y por el enfoque de HMMs [Avilés-Arriaga et al., 2006]. Para las gramáticas aprendidas usando 2 secuencias, la precisión es de 95.17 %, mientras que para HMMs es de 94.85 %. Para las gramáticas aprendidas usando 10 secuencias, la precisión es de 97.34 % y HMMs logra 97.56 %. En ambos casos, el desempeño de FOSeq es similar al logrado por HMMs. Estos resultados son prometedores puesto que HMMs ha sido la técnica generalmente usada para clasificación de ademanes. Una ventaja de la clasificación usando gramáticas es que éstas describen explícitamente la estructura del ademán a diferencia de HMMs que se concentran en la clasificación.

2 secuencias de entrenamiento		10 secuencias de entrenamiento	
FOSeq	HMMs	FOSeq	HMMs
95.17 %	94.85 %	97.34 %	97.56 %

Tabla 6.8: Precisión: Comparación entre FOSeq y HMMs

La Tabla 6.9 muestra la matriz de confusión para 2 secuencias de entrenamiento. Los mejores resultados los obtuvieron los ademanes `acercar` y `giro-hacia-la-izquierda` con 100.00 % de precisión. El ademán que tiene más errores de clasificación es `apuntar`, con 82.00 % de precisión. De los ejemplos del ademán `apuntar` mostrados a las gramáticas, el 7.93 % fueron clasificados como `izquierda`, el 7.24 % como `acercar` y el 2.76 % como `atención`. Estos resultados coinciden con los obtenidos en [Avilés-Arriaga et al., 2006], en donde los ademanes que mostraron mayor confusión fueron: `apuntar`, `detenerse`, `acercar` y `atención`.

La Tabla 6.10 muestra los resultados de clasificación para gramáticas aprendidas con 10 secuencias de entrenamiento. Los ademanes con mayor precisión son `izquierda`, `derecha`, `giro-hacia-la-derecha` y `apuntar` (100 %). Al igual que en la matriz anterior, se muestra

		Predicción									
		1	2	3	4	5	6	7	8	9	Total
Clase real	1	**97.00**							3.00		300
	2		**91.03**	3.10	5.86						290
	3			**100.00**							290
	4	2.76	7.24	7.93	**82.07**						290
	5	0.67				**97.67**				1.67	300
	6	2.07					**93.79**			4.14	290
	7					2.07		**96.55**		1.38	290
	8								**100.00**		290
	9					0.34	1.38			**98.28**	290
	Total	307	285	322	255	300	276	280	299	306	2630

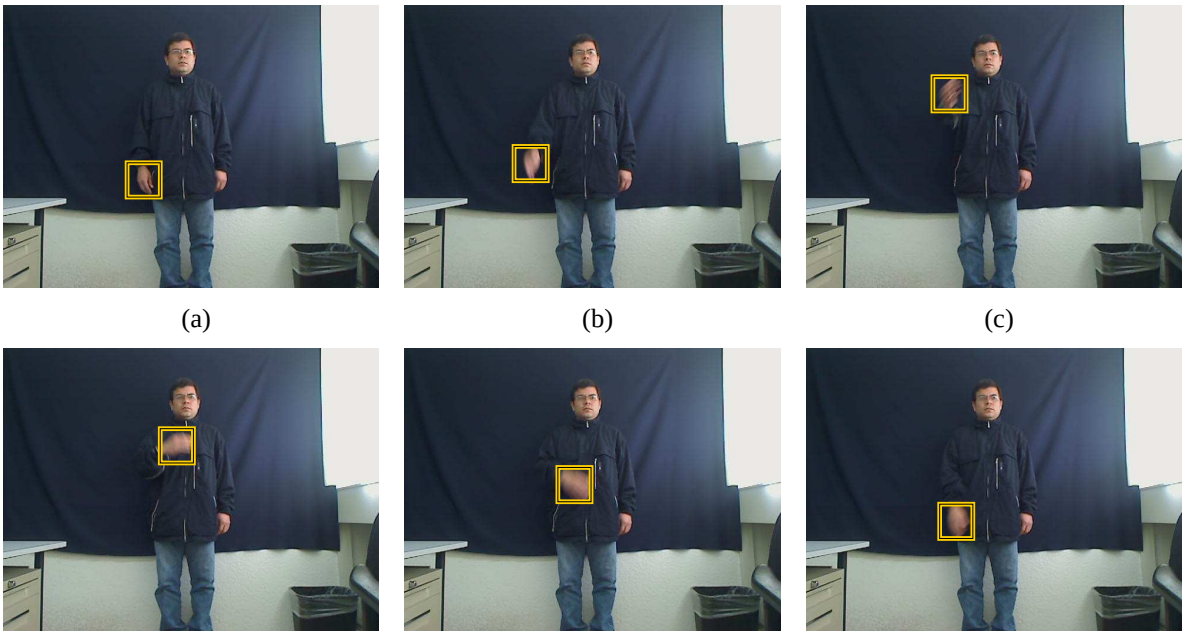
Tabla 6.9: Matriz de confusión para dos secuencias de entrenamiento. Clases: 1) `atención`, 2) `acercar`, 3) `izquierda`, 4) `apuntar`, 5) `derecha`, 6) `detenerse`, 7) `giro-hacia-la-izquierda`, 8) `giro-hacia-la-derecha`, 9) `saludar`. La última columna indica el número total de secuencias de cada ademán. El último renglón corresponde al número de secuencias clasificadas como la clase indicada en la columna.

que el ademán `apuntar` obtuvo la menor precisión ($85.17\,\%$), del cual $13.10\,\%$ de los ejemplos fueron clasificados como `acercarse`. El porcentaje global alcanzado es de $97.34\,\%$. De ambas matrices de confusión se observa que el incremento en el número de secuencias para aprendizaje de la gramática mejora la precisión global de reconocimiento. El aprendizaje y resultados correspondientes a ésta sección se describen también en [Vargas-Govea y Morales, 2009].

		Predicción									
		1	2	3	4	5	6	7	8	9	Total
Clase real	1	**99.33**							0.67		300
	2		**97.24**		2.76						290
	3			**100.00**							290
	4		13.10	1.72	**85.17**						290
	5					**100.00**					300
	6	1.03					**97.59**			1.38	290
	7					2.07		**96.55**		1.38	290
	8								**100.00**		290
	9									**100.00**	290
	Total	301	320	295	255	306	283	280	292	298	2630

Tabla 6.10: Matriz de confusión para 10 secuencias de entrenamiento. Clases: 1) `atención`, 2) `acercar`, 3) `izquierda`, 4) `apuntar`, 5) `derecha`, 6) `detenerse`, 7) `giro-hacia-la-izquierda`, 8) `giro-hacia-la-derecha`, 9) `saludar`. La última columna indica el número total de secuencias de cada ademán. El último renglón corresponde al número de secuencias clasificadas como la clase indicada en la columna.

Una representación explícita permite explorar las diferencias entre los ademanes. Podemos analizar las reglas para los ademanes parecidos, en particular los que presentaron mayor confusión en los experimentos de clasificación: `apuntar` y `acercar`. Lo esperado sería que las gramáticas de ambos presentaran reglas semejantes. La Tabla 6.11 muestra la gramática que describe al ademán `apuntar`. La Tabla 6.12 muestra la gramática para el ademán `acercar`. Se puede observar que las reglas de `apuntar`: $R1$, $R4$, $R8$, $R2$ y $R3$ son iguales que las reglas de `acercar`: $R1$, $R5$, $R3$ y $R2$ (en negritas). Las reglas muestran los atributos de movimiento y postura que son comunes a ambos ademanes. Por ejemplo, la regla $R1$ dice que en alguna parte de la secuencia, la mano está sobre el torso pero no a la altura del rostro. Esta regla forma parte de la regla $R2$. En la secuencia S de `apuntar`, $R2$ es el séptimo elemento, de un total de 26, mientras que en `acercar`, $R2$ es el sexto elemento de un total de 25. La mano sobre el torso ocurre aproximadamente a la misma altura de la secuencia en ambos ademanes. Las Figuras 6.27 (a), (b) y (c) del ademán `acercar` son muy parecidas a las Figuras 6.27 (a), (b) y (c) 6.28 del ademán `apuntar`. Las reglas pueden interpretarse como sub-ademanes que se encuentran a lo largo de la secuencia y que pueden encontrarse en ademanes diferentes.

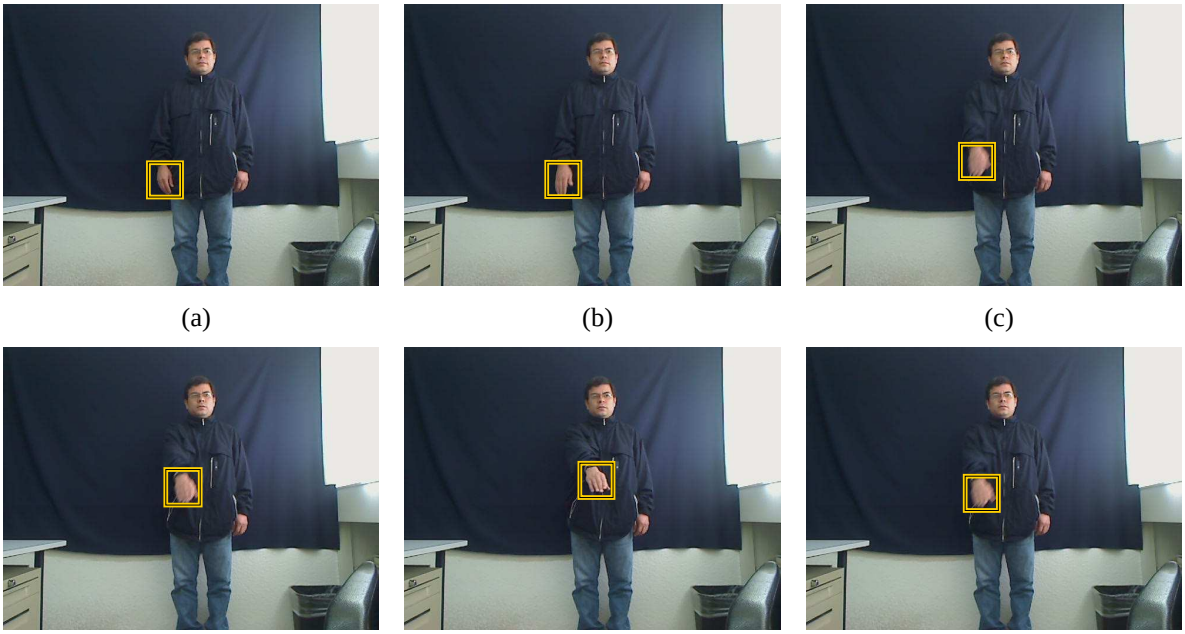


(a) (b) (c)

(d) (e) (f)

Figura 6.27: Ademán `acercar`

El aprendizaje de gramáticas relacionales para reconocimiento de ademanes es de utilidad para encontrar similaridades entre ademanes distintos y son competitivas con HMMs. Aunque



(a) (b) (c)

(d) (e) (f)

Figura 6.28: Ademán `apuntar`

la reproducción de secuencias de ademanes no fue probada, los experimentos en el dominio de navegación muestran que las gramáticas aprendidas tienen un buen desempeño como programas de control.

$S \rightarrow R9$,$R4$,$R5$,$R4$,$tamaño$(`inc`),$forma$(`vertical`),$R2$,$hmov$(`ninguno`),$R11$, $hmov$(`derecha`),$vmov$(`subiendo`),$R5$,$R10$,$hmov$(`derecha`),$R11$,$R10$, $R12$,$forma$(`inclinada`),$R2$,$R8$,$tamaño$(`dec`),$R6$,$R1$,$R12$,$R7$
R1 $\rightarrow$ ***en_rostro(Estado,***`no`***),sobre_torso(Estado,***`si`***)***
R4 $\rightarrow$ ***hmov(Estado,***`izquierda`***),vmov(Estado,***`subiendo`***)***
R8 $\rightarrow$ ***hmov(Estado,***`derecha`***),vmov(Estado,***`bajando`***)***
R3 $\rightarrow$ ***forma(Estado,***`horizontal`***),R2***
R2 $\rightarrow$ ***derecha(Estado,***`no`***),R1***
$R10 \rightarrow hmov(Estado$,`izquierda`$),vmov(Estado$,`bajando`$),R5$
$R11 \rightarrow vmov(Estado$,`subiendo`$),tamaño(Estado$,`dec`$),R3$
$R12 \rightarrow R8$,$tamaño(Estado$,`inc`$)$
$R5 \rightarrow tamaño(Estado$,`inc`$),R3$
$R9 \rightarrow R4$,$tamaño(Estado$,`dec`$),R7$
$R6 \rightarrow forma(Estado$,`vertical`$), derecha(Estado$,`si`$)$
$R7 \rightarrow R6\ arriba\_rostro(Estado$,`no`$), sobre\_torso(Estado$,`no`$)$

Tabla 6.11: Gramática del ademán `apuntar`

$S \rightarrow R5$,$tama\tilde{n}o$(dec),$forma$(vertical),$R12$,$R10$,$R2$,$R7$,$R11$,$R13$, $vmov$(bajando), $tama\tilde{n}o$(inc),$R11$,$R6$,$R13$,$vmov$(ninguno),$tama\tilde{n}o$(inc), $R14$,$R6$,$forma$(horizontal),$R8$,$R3$,$tama\tilde{n}o$(dec),$forma$(horizontal), $derecha$(si),$arriba\_rostro$(no),$sobre\_torso$(no)
R1 $\rightarrow$ ***en_rostro(Estado,***no***),sobre_torso(Estado,***si***)***
R5 $\rightarrow$ ***hmov(Estado,***izquierda***),vmov(Estado,***subiendo***)***
R3 $\rightarrow$ ***hmov(Estado,***derecha***),vmov(Estado,***bajando***)***
R2 $\rightarrow$ ***forma(Estado,***horizontal***),derecha(Estado,***no***),R1***
$R11 \rightarrow R2$,$hmov(Estado$,izquierda$)$,$vmov(Estado$,ninguno$)$, $tama\tilde{n}o(Estado$,dec$)$
$R12 \rightarrow R10$,$forma(Estado$,horizontal$)$
$R13 \rightarrow R2$,$hmov(Estado$,ninguno$)$
$R6 \rightarrow R4$,$tama\tilde{n}o(Estado$,inc$)$
$R7 \rightarrow R5$,$tama\tilde{n}o(Estado$,inc$)$
$R8 \rightarrow derecha(Estado$,si$)$,$R1$
$R9 \rightarrow R4$,$tama\tilde{n}o(Estado$,dec$)$
$R10 \rightarrow R8$,$R7$
$R14 \rightarrow R6$,$R9$
$R4 \rightarrow R2$,$R3$

Tabla 6.12: Gramática del ademán acercar

6.6 Análisis de sensibilidad

Para efectuar el análisis de sensibilidad se tomaron en cuenta dos factores, por un lado la cantidad de los ejemplos y por otro la cantidad de conocimiento del dominio que se agrega para el aprendizaje.

6.6.1 Ejemplos

El número de ejemplos es un factor que influye en la precisión de las reglas o PTRs inducidos. Esto es común para el área de aprendizaje en donde la cantidad y calidad de los ejemplos afectan directamente a la calidad del concepto que se quiere aprender. Del número total de ejemplos con los que se realizó el aprendizaje de los PTRs de navegación, se efectuaron pruebas para observar el efecto de la reducción de ejemplos en el PTR aprendido. La Tabla 6.13 muestra el nombre del PTR, sus acciones, el número de ejemplos por acción

(#ej/acc) y el número total de ejemplos(#ej total). Se muestra también el número mínimo de ejemplos tanto por acción (#ej mínimo/acc) como en total (#ej mínimo total) con los que se logró aprender el mismo PTR. Sin embargo, el número mínimo de ejemplos es con respecto a los ejemplos obtenidos para los experimentos mostrados en esta tesis, no se puede garantizar que para nuevos ejemplos y cambios en predicados o conocimiento del dominio, el número mínimo de ejemplos sea el mismo. Agregar más ejemplos no mejora la precisión de los PTRs aprendidos, únicamente se generan reglas redundantes.

El número de ejemplos por acción en cada PTR es parecido, balanceado, por ejemplo, para el PTR *deambular* cada acción tiene aproximadamente 300 ejemplos. Se hizo de esta manera porque cuando existe alguna acción con muy pocos ejemplos en relación a las demás, generalmente no se induce un predicado para aplicar la acción faltante. Se observa que el PTR para orientarse tiene más ejemplos que el resto (2982) ésto se debe a que en un principio se creyó necesario agregar ejemplos que cubrieran los 4 cuadrantes. Se capturaron ejemplos por cada grado de giro y el robot iniciando en muchas posiciones distintas por lo que se generaron más ejemplos que para el resto de los PTRs.

En la tabla se muestra que el PTR para ir-a-un-punto puede ser aprendido a partir de 8 secuencias en vez de las 12 originales. Las 8 secuencias corresponden a trayectorias que cubren las acciones que el PTR probado necesita. Las 4 secuencias que no fueron necesarias cubrían casos redundantes. Las secuencias fueron generadas con la intención de cubrir los casos necesarios sin considerar que existieran casos redundantes.

Para el aprendizaje de gramáticas de ademanes, cuando el entrenamiento se realizó con 1 secuencia, la precisión fue de 95.17 % mientras que para 10 secuencias, se obtuvo 97.34 %. Al aumentar el número de ejemplos, la precisión mejoró.

6.6.2 Conocimiento del dominio

El conocimiento del dominio es un factor crítico en el aprendizaje, sin embargo, agregar una gran cantidad de predicados de conocimiento del dominio no necesariamente genera un mejor aprendizaje. Al igual que con los ejemplos el factor mas importante en la relevancia o calidad, mas que la cantidad. Si el conocimiento dado es, éste aparecerá en el predicado aprendido, pero si es irrelevante, no aparecerá. Por el contrario, si se omite información fundamental, difícilmente se inducirá un PTR que controle al robot de la forma esperada.

En esta sección se muestra el efecto de eliminar conocimiento del dominio del predicado objetivo para aprender a deambular. Se eliminó un predicado correspondiente a conocimiento del dominio y se dio como entrada a ALEPH, usando los mismos ejemplos que en aprendizaje

PTR	acción	#ej/acc	#ej total	#ej mínimo/acc	#ej mínimo total
Deambular	*girar_derecha*	317	979	57	188
	girar_izquierda	325		60	
	avanzar	337		71	
Orientarse	*girar_derecha*	993	2982	23	71
	girar_izquierda	1194		30	
	detenerse	895		18	
Salir de trampa	*girar_derecha*	235	952	110	428
	girar_izquierda	235		120	
	detenerse	245		100	
	retroceder	237		98	
Seguir	*girar_derecha*	117	619	61	313
	girar_izquierda	117		58	
	avanzar	147		69	
	retroceder	120		71	
	detenerse	118		54	
Ir a un punto			12 secuencias		8 secuencias

Tabla 6.13: Análisis de sensibilidad al número de ejemplos para PTRs de navegación básicos

del predicado objetivo original. Se observó que al eliminar conocimiento del dominio que proporcione informacin fundamental no se aprende un predicado que realice la tarea esperada. Por ejemplo, el predicado *obst-mas-cercano* es necesario porque obtiene la distancia y el ángulo hacia la marca más cercana y a partir de esos valores se aprenden los rangos para ejecutar las acciones.

Otros predicados no son tan críticos, como es el caso de los predicados *mayor-que* y *menor-que*. Puede eliminarse alguno de los dos y se logra aprender un PTR con el cual el robot puede deambular. Depende de cuál predicado se elimine, no de la cantidad, el aprender un predicado que permita que el robot deambule sin chocar. El predicado objetivo del cual se aprendió el PTR para deambular es el siguiente:

$$
\begin{aligned}
&deambular(+estado, \#acc).\\
&\quad zona{-}frontal(+estado, \#valorzona),\\
&\quad obst{-}mas{-}cercano(+estado, +marca, +distancia, +angulo),\\
&\quad menor{-}igual(+angulo1, \#valorangulo1),\\
&\quad mayor{-}que(+angulo2, \#valorangulo2).
\end{aligned}
$$

donde *deambular(+estado,#acc)* es la cabeza del predicado a aprender, *+estado* es el estado recibido, y *#acc* es la posible acción a ejecutar. Las posibles acciones pueden ser *girar-*

izquierda, *girar-derecha* y *girar-avanzar*. Los posibles predicados son: *zona-frontal*, *obst-mas-cercano*, *menor-igual* y *mayor-que*. El predicado *zona-frontal* indica si existe un obstáculo al frente del robot. El predicado aprende los rangos de distancia y ángulo a los que el usuario considera como obstáculo las marcas que ve. Para aprender los rangos de distancia y ángulo se agregó como conocimiento del dominio los predicados *mayor-igual* y *menor-que*. A continuación se describen los efectos en el PTR aprendido al eliminar predicados del conocimiento del dominio.

- Si se elimina el predicado *mayor-que*, el PTR aprendido es muy parecido al original. La diferencia es que en el predicado para ejecutar la acción `izquierda`, el valor del Ángulo es -0.5 mientras que en el PTR original el valor es de -1.5.

```
deambular(Estado, avanzar) :-
    zona-frontal(Estado, sin-obstaculo).
deambular(Estado, izquierda) :-
    zona-frontal(Estado, con-obstaculo),
    obst-mas-cercano(Estado, Marca, Distancia, Angulo),
    menor-igual(Angulo, -0.5).
deambular(Estado, derecha) :-
    zona-frontal(Estado, con-obstaculo).
```

- Si se elimina el predicado *menor-que*, el rango de valores se aprende para ejecutar la acción `derecha`. A una orientación mayor a 10.56 grados hacia la marca más cercana, el robot girará hacia la derecha, sin embargo, ésto lo hará sin importar la distancia. No siempre la marca más cercana está a una distancia de riesgo, si no es así, el robot girará innecesariamente. Si no hay una marca hacia la derecha del robot, pero sí existe un obstáculo cerca entonces el robot girará hacia la izquierda. El robot avanzará si no existe un obstáculo en la *zona frontal*.

```
deambular(Estado, derecha) :-
    obst-mas-cercano(Estado, Marca, Distancia, Angulo),
    mayor-que(Angulo, 10.56).
deambular(Estado, izquierda) :-
    zona-frontal(Estado, con-obstaculo).
deambular(Estado, avanzar) :-
    zona-frontal(Estado, sin-obstaculo).
```

- Si se elimina el predicado *zona-frontal*, se aprende un PTR que indica si hay que girar hacia la `izquierda` o hacia la `derecha` pero no indica si existe o no un obstáculo cerca. El robot girará sin importar la distancia hacia la que está el obstáculo. En este caso, para ambas acciones `izquierda` y `derecha`, el robot girará innecesariamente cuando la marca percibida no sea un riesgo. Al no existir una condición de distancia, el riesgo de chocar aumenta pues la acción `avanzar` la ejecutará en caso de que ninguna de las condiciones para girar se cumplan.

```
deambular(Estado, izquierda) :-
    obst-mas-cercano(Estado, Marca, Distancia, Angulo),
    menor-igual(Angulo, 88.99).
deambular(Estado, izquierda) :-
    obst-mas-cercano(Estado, Marca, Distancia, Angulo),
    menor-igual(Angulo, -2.51).
deambular(Estado, avanzar).
```

- Si se elimina el predicado para la marca mas cercana, el PTR aprendido indica girar hacia `izquierda` o `derecha` si existe un obstáculo, pero si existe, la acción es `avanzar`. No se conoce la orientación del obstáculo por lo que por ser la primer regla, el robot siempre girará hacia la `izquierda` cuando exista un obstáculo en la zona frontal. Si el obstáculo está a la izquierda, el robot siempre chocará.

```
deambular(Estado, izquierda) :-
    zona-frontal(Estado, con-obstaculo).
deambular(Estado, izquierda) :-
    zona-frontal(Estado, con-obstaculo).
deambular(Estado, avanzar) :-
    zona-frontal(Estado, sin-obstaculo).
```

Los predicados críticos en el aprendizaje del PTR para *deambular* son *zona-frontal* y *obst-mas-cercano*. Si estos predicados se omiten, el comportamiento del robot se degrada por la falta de información fundamental para la tarea que se quiere aprender.

En general, con la mayoria de los sistemas de aprendizaje computacional, es mas importante la calidad/relevancia de los datos y conocimiento que la cantidad. Se requiere cierta cantidad minima para hacer las inferencias adecuadas, pero si no se tienen los datos o conocimiento relevantes, el sistema no es capaz de generar conceptos aceptables. Como normalmente el usuario no sabe de antemano qué es lo relevante, normalmente se proporciona más de lo necesario (ejemplos y conocimiento), que es lo que se hizo en la tesis, se deja que el sistema de aprendizaje seleccione lo relevante.

6.7 Resumen

En este capítulo se mostraron los resultados de los experimentos de PTRs para navegación en simulación y en un robot real. Se probaron también las gramáticas de ademanes como clasificadores de nuevas secuencias de ademanes.

- Se realizaron experimentos en simulación para el conjunto completo de PTRs aprendidos y se obtuvo su precisión: ***deambular*** (86.67 %),*orientarse*(100.00 %), ***salir de trampa*** (80.00 %), ***seguir*** objeto (97.50 %), ***deambular*/*orientarse*** (ir a un punto) (86.67 %), ***deambular*/*orientarse*/*salir de trampa*** (93.33 %), ***seguir*/*deambular*** (90.00 %), ***seguir*/*deambular*/*salir de trampa*** (100.00 %) y visitar 5 lugares (96.67 %).

- En el robot de servicio se probaron dos PTRs, obteniéndose su precisión: ir-a-un-punto (85.00 %) y ***seguir*** a una persona (90.00 %).

- Se integraron los PTRs a una plataforma de robot de servicio. Los PTRs se probaron como módulo de navegación para un robot móvil que ejecuta tareas de servicio (*e.g.*, mensajería, búsqueda de objetos, seguimiento de personas).

- Se comparó el desempeño de los PTRs con un enfoque deliberativo. Los PTRs dan al robot la ventaja de reaccionar ante eventos inesperados, en ambientes desconocidos. Con DELIB se genera la política para un ambiente conocido pero requiere generar una nueva política si hay cambios en él. El uso combinado del enfoque deliberativo y reactivo muestra resultados más robustos particularmente en trayectos donde los espacios son reducidos o saturados de objetos. Sin embargo, cuando los espacios son abiertos y se le presenta un obstáculo, puede generar trayectorias más complicadas.

- Se probaron las gramáticas de ademanes aprendidas como clasificadores de nuevas secuencias. Se mostraron los resultados de clasificación utilizando conjuntos de secuencias de prueba.

- Se analizaron las reglas para los ademanes parecidos, en particular los que presentaron mayor confusión en los experimentos de clasificación: `apuntar` y `acercar`. En ambas gramáticas se encontraron reglas idénticas. La representación explícita es de gran utilidad para encontrar similaridades entre ademanes distintos a diferencia de otros enfoques que no muestran este tipo de información.

- En ambos dominios, navegación robótica y reconocimiento de ademanes, las gramáticas permiten identificar sub-tareas, en el caso de las trayectorias del primero y sub-ademanes en el caso del segundo.

- Se redujo el número de ejemplos en el aprendizaje de los PTRs para observar el efecto del número de ejemplos en el PTR resultante. Se observó que el número de ejemplos por acción y agregar el conocimiento del dominio correcto son factores que influyen de forma determinante en el aprendizaje y precisión del PTR inducido.

Los resultados muestran que las gramáticas como PTRs permiten controlar a un robot para realizar tareas de navegación tanto en simulación como en ambientes reales. Por otro lado, como clasificadores de secuencias de ademanes son competitivas con respecto a otros enfoques tradicionales con la ventajas de una representación explícita. En el siguiente capítulo se presentan las conclusiones y el trabajo futuro que puede derivarse de esta tesis.

7

Conclusiones y trabajo futuro

7.1 Sumario

Conforme se desarrollan los robots de servicio surge la necesidad de facilitar su programación. La programación tradicional es una tarea que puede consumir mucho tiempo y sólo la puede realizar un especialista. Sin embargo, con la tendencia a incorporar robots móviles a la vida cotidiana y a realizar tareas domésticas y de servicio se necesitan métodos para programarlos sin ser expertos.

Programar guiando un robot abre nuevas posibilidades pues la persona que "entrena" al robot sólo se preocupa por definir la tarea y de conducir al robot a hacerla. Es una forma de mostrarle al robot qué hacer en vez de decirle explícitamente cómo hacerlo. En esta tesis se desarrolla una metodología de aprendizaje que es un paso adelante para hacer realidad este panorama.

El principal objetivo de esta tesis es desarrollar una plataforma de aprendizaje relacional de tareas para un robot móvil. A partir de trazas obtenidas mientras el usuario guía a un robot con la palanca de control (*joystick*), el robot aprende habilidades para realizar las tareas.

En el proceso de aprendizaje se identificaron los siguientes objetivos específicos:

- Desarrollar mecanismos que permitan manejar los cientos de datos de bajo nivel recibidos de los sensores en cada instante para construir una representación relacional a partir de información significativa.

- Aprender habilidades, en este caso expresadas mediante PTRs a partir de una representación relacional usando trazas generadas por una persona.

- Desarrollar un algoritmo cuya salida sean programas lógicos que expresen la estructura jerárquica de secuencias de acciones.

Las principales contribuciones de esta tesis son:

- Representación de alto nivel mediante una transformación de información de bajo nivel (sensores). Para lograr este objetivo se utiliza un proceso de identificación de marcas naturales del ambiente a partir de la información de los sensores del robot. Esta información junto con conocimiento del dominio adicional se usa para generar una representación relacional que facilita el uso de algoritmos de aprendizaje.

  La mayoría de los trabajos del estado del arte manejan datos crudos, representaciones manualmente construidas y a un solo nivel. En esta tesis se generan automáticamente distintos niveles de abstracción de la información. Uno de los aspectos fundamentales del aprendizaje es la representación. A partir de datos crudos de los sensores, se produce una representación de alto nivel mediante un proceso de identificación de marcas naturales que posteriormente se transforma a secuencias de PTRs básicos.

- Algoritmo para aprendizaje de conceptos y PTRs básicos usando clonación y programación lógica inductiva. A partir de ejemplos generados por el usuario se aprenden conceptos que identifican zonas del ambiente. Los PTRs que se aprenden utilizan ésos conceptos como conocimiento del dominio, de esta manera, si el robot está por ejemplo en una zona identificada como "libre", el PTR puede utilizar este concepto para avanzar sin peligro. Para aprender PTRs se usan trazas que se obtienen al guiar al robot, se transforman a una representación de alto nivel y se dan como entrada junto con posible conocimiento del dominio (e.g. los conceptos aprendidos) a un sistema de *ILP*. El resultado es un conjunto de PTRs básicos que permiten al robot realizar la tarea mostrada por el usuario.

  En la mayoría de las aplicaciones de PTRs estos se definen manualmente. En los casos donde los PTRs sson aprendidos se prueban sólo en simulación con sensores muy limitados y no son jerárquicos. La clonación había sido usada en robótica móvil usando algoritmos proposicionales que dificultan el aprendizaje de habilidades combinadas.

  En esta tesis se desarrolla una metodología de aprendizaje relacional y clonación para aprendizaje de PTRs de navegación para robots móviles. Se aprenden conceptos y PTRs básicos usando *ILP*. En contraste con los trabajos previos se probaron tanto en simulación como en un robot real.

- Algoritmo para aprendizaje de PTRs compuestos. Un PTR compuesto está formado por dos o más PTRs básicos. La idea es identificar en las trazas obtenidas al realizar una tarea con PTRs previamente aprendidos con los cuales se puedan construir automáticamente PTRs compuestos. Se presenta un algoritmo que a partir de secuencias aprende programas lógicos que representan la estructura jerárquica de la secuencia de entrada, y se usaron: (i) como programas lógicos en la forma de PTRs compuestos y (ii) como clasificadores de secuencias de ademanes.

  La idea de aprender PTRs jerárquicos nos llevó al aprendizaje de gramáticas. El aprendizaje de gramáticas se enfoca principalmente al área de procesamiento de lenguaje natural por lo que los algoritmos existentes son muy específicos y no aplicables a otros dominios. Para reconocimiento de acciones existen trabajos que incorporan gramáticas construidas manualmente como complemento de otros métodos. Sin embargo, en el estado del arte no se encontró la inducción de gramáticas a partir de secuencias de acciones que se usen como programas y para clasifición.

  En esta tesis se presenta un algoritmo (FOSeq) que aprende gramáticas de cláusulas definidas (DCGs) a partir de secuencias de pares estado-acción común a varios dominios. Un proceso de generalización permite construir una gramática a partir de varias que aporten elementos diferentes.

7.2 Conclusiones

- La generación automática de una representación relacional a partir de datos crudos a diferentes niveles es una contribución y abre posibilidades de utilizar diferentes algoritmos de aprendizaje al reducir el volumen de datos que los sensores producen. Este enfoque relacional permite que los PTRs aprendidos en un ambiente puedan ser utilizados en nuevos ambientes con características similares.

- El aprendizaje automático de PTRs básicos a partir de una representación relacional utilizando trazas generadas por una persona facilita la programación de un robot. El programador no necesita experiencia en la programación de robots para que el robot pueda realizar sub-tareas

- El aprendizaje de PTRs jerárquicos utilizando como base los PTRs básicos permite expresar de forma jerárquica las acciones del robot. Ésto es importante porque es posible

expresar sub-tareas en el caso de navegación o sub-ademanes en el caso de la clasificación de ademanes. La representación explícita de las jerarquías permiten analizar similitudes y diferencias entre una tarea/sub-tarea y otras. El aprendizaje de gramáticas es una contribución relevante porque a diferencia de otros enfoques, las gramáticas inducidas por FOSeq pueden tener dos aplicaciones: ser usadas para ejecutar una tarea particular y para clasificar nuevas secuencias.

- La inducción de gramáticas se probó en el aprendizaje de PTRs de navegación jerárquicos y para generar gramáticas de ademanes para clasificación. En ambas aplicaciones se obtuvieron buenos resultados: los PTRs jerárquicos fueron probados tanto en simulación como en un robot real. Su naturaleza reactiva permite reaccionar a eventos inesperados y el enfoque relacional permite que puedan usarse en ambientes diferentes al que fue entrenado. En la aplicación de clasificación de ademanes los resultados obtenidos son competitivos en comparación con sistemas del estado del arte. Las gramáticas de ademanes aprendidas son interpretables y capaces de mostrar sub-ademanes.

  Se obtuvieron resultados competitivos en comparación con sistemas del estado del arte. Una ventaja de las gramáticas es que proporcionan reglas explícitas del ademán que con otros enfoques no se obtiene.

- Dos aspectos críticos son el conocimiento del dominio y los ejemplos. El conocimiento del dominio tiene como ventaja permitir al usuario intervenir en el proceso de aprendizaje y sus resultados. Proporciona flexibilidad al usuario si tiene conocimiento parcial, intuiciones o expectativas sobre la hipótesis a ser inducida. Sin embargo, si no se tiene, el espacio de búsqueda será mayor, puede necesitar más ejemplos y tiempo y en el peor de los casos puede no tener éxito. El número y la calidad de los ejemplos son también determinantes en los resultados. Pueden darse muchos ejemplos bajo situaciones parecidas, originando baja precisión al presentarse al robot nuevos estados. Por el contrario, si se dan pocos ejemplos bajo situaciones diversas, la generalización puede ser deficiente. Con un balance adecuado entre el número de ejemplos y diversidad de situaciones se logra aprender reglas generales que permiten al robot realizar la tarea. Sin embargo, determinar este balance no siempre es sencillo.

- En el aprendizaje de PTRs mediante clonación, al proporcionar ejemplos positivos y negativos como parte del aprendizaje, el tiempo de inducción es menor que por ejemplo, en aprendizaje por refuerzo relacional (RRL). Por otro lado, el formalismo teleo-reactivo

de los PTRs permite aplicar acciones continuas y manejar metas/sub-metas dentro de la misma representación. El formalismo teleo-reactivo da a los PTRs manejo de metas y de acciones continuas que otros enfoques pueden extender pero tendrían que agregar mecanismos para lograrlo.

- La inducción de gramáticas se probó en el aprendizaje de PTRs de navegación jerárquicos y para generar gramáticas de ademanes para clasificación. En ambas aplicaciones se obtuvieron buenos resultados: los PTRs jerárquicos fueron probados tanto en simulación como en un robot real. Su naturaleza reactiva permite reaccionar a eventos inesperados y el enfoque relacional permite que puedan usarse en ambientes diferentes al que fue entrenado. En la aplicación de clasificación de ademanes los resultados obtenidos son competitivos en comparación con sistemas del estado del arte. Las gramáticas de ademanes aprendisas son interpretables y capaces de mostrar sub-ademanes. Una ventaja de las gramáticas es que proporcionan reglas explícitas del ademán que con otros enfoques no se obtienen.

7.3 Trabajo futuro

Existen distintas líneas de trabajo futuro que pueden derivarse de los resultados de esta tesis. Algunas tienen por objetivo cubrir las limitaciones, y otras son extensiones que no corresponden al alcance definido para este trabajo. Entre éstas:

- Establecimiento de diferentes criterios de evaluación de comportamientos. No existen medidas de evaluación estándar para comportamientos o habilidades. La medición usando solamente precisión no considera aspectos de la tarea general como por ejemplo el tipo de ambiente en el cual el comportamiento es evaluado. La evaluación por precisión puede ser muy sesgada si no se considera la facilidad o dificultad del ambiente de pruebas.

- Aprendizaje con retroalimentación. Actualmente el aprendizaje se realiza con ejemplos leídos de un archivo. Si al ejecutar la tarea el robot desconoce la acción para un estado, ejecuta una acción por omisión hasta que encuentra un estado conocido. Para poder mejorar el aprendizaje conforme el robot descubre estados se podrían incorporar mecanismos para dar al robot retroalimentación y que pueda agregar nuevos ejemplos para el proceso de aprendizaje. Una posibilidad es incorporar interacción humana, como comandos de voz para indicar al robot cuál es la acción que debe realizar.

- Aprendizaje de PTRs para resolver situaciones de conflicto. Actualmente si el PTR no puede cumplir con la meta por caer en un ciclo o quedar atrapado, es necesario que una persona "libere" al robot. El robot no es capaz de reconocer que está en un ciclo. Una posible extensión es aprender PTRs para que el robot identifique que está en una situación difícil y que aprenda estrategias para liberarse por sí mismo, o en su defecto que determine que no puede salir y se apague. Al ser un enfoque reactivo, el robot reacciona de acuerdo a lo que percibe solamente en ése instante. Para este tipo de PTRs se necesita incorporar un mecanismo de memoria que le permita al robot identificar que ya pasó por los mismos lugares.

- Aprendizaje de valores de parámetros. Actualmente, el aprendizaje de PTRs asocia una acción a un estado. Se codificaron manualmente variaciones en la velocidad para el PTR ***seguir*** dependiendo de la cercanía o lejanía del objeto a seguir con lo que se mejora el comportamiento. Sin embargo, parámetros como la velocidad de desplazamiento y la velocidad angular con las que una persona guía al robot también pueden aprenderse. Se podrían realizar tareas con distintos estilos de ejecución, ya sea para urgencias o de forma normal.

- Ampliación de la representación del ambiente. La representación del ambiente se basa en paredes, esquinas y discontinuidades. Estos elementos proporcionan información suficiente para realizar tareas de navegación. Sin embargo, para diversificar el tipo de tareas es necesario incorporar un proceso de identificación de más elementos que enriquezca la representación con más información. Se puede pensar en una jerarquía de lugares donde los elementos básicos son los mínimos que construyen el ambiente, en este caso, las paredes, esquinas y discontinuidades. El siguiente nivel estaría formado por los elementos construidos a partir de elementos básicos como habitaciones, pasillos, puertas, estancias. El siguiente nivel correspondería a la identificación por el contenido de las habitaciones, por ejemplo, cocina, baño, sala. Un nivel superior sería aquél en el que se identifiquen los lugares por el tipo de habitaciones por ejemplo, si el lugar tiene cocina, baño, recámaras y sala, lo más probable es que se trate de una casa o departamento mientras que si el lugar tiene oficinas y baño, es más posible que se trate de un sitio de trabajo de un instituto o de una empresa.

- Aprendizaje de jerarquías con mas niveles. Los experimentos se limitaron al aprendizaje de sub-tareas de navegación con pocos niveles. Al extender la representación del ambiente y aprender tareas más complicadas se esperaría que las gramáticas apren-

didas de las secuencias mostraran una descomposición a mayor detalle de la tarea o sub-tarea aprendida.

- Creación de nuevos predicados. El aprendizaje a partir de secuencias se limita a predicados dados o aprendidos con la guía del usuario. El robot puede estar en una situación que desconozca y sería muy útil que el robot fuera capaz de ir recolectando esos estados desconocidos y efectuando el proceso de aprendizaje mientras se va desplazando. El usuario podría intervenir indicándole la acción a efectuar.

- Mejora del algoritmo de generalización en el aprendizaje de gramáticas. El algoritmo de generalización actual considera solamente predicados con dos argumentos. Es necesario extender y hacer más flexible el algoritmo para que pueda generalizar predicados por ejemplo, con un mayor número de argumentos.

Referencias

[Adriaans et al., 2000] Adriaans, P. W., Trautwein, M., y Vervoort, M. (2000). Towards high speed grammar induction on large text corpora. En *SOFSEM '00: Proceedings of the 27th Conference on Current Trends in Theory and Practice of Informatics*, pags. 173–186, London, UK. Springer-Verlag.

[Agrawal y Srikant, 1994] Agrawal, R. y Srikant, R. (1994). Fast algorithms for mining association rules. En Bocca, J. B., Jarke, M., y Zaniolo, C., editores, *Proceedings of 20th International Conference of Very Large Data Bases, VLDB*, pags. 487–499. Morgan Kaufmann.

[Amit y Mataric, 2002] Amit, R. y Mataric, M. J. (2002). Learning movement sequences from demonstration. En *ICDL '02: Proceedings of the 2nd International Conference on Development and Learning*, pags. 12–15, Cambridge, MA.

[Avilés-Arriaga et al., 2006] Avilés-Arriaga, H. H., Sucar, L. E., y Mendoza, C. E. (2006). Visual recognition of similar gestures. *18 International Conference on Pattern Recognition ICPR 2006*, 1:1100–1103.

[Bain y Sammut, 1999] Bain, M. y Sammut, C. (1999). A framework for behavioural cloning. En *Machine Intelligence 15*, pags. 103–129. Oxford University Press.

[Benson y Nilsson, 1995] Benson, S. y Nilsson, N. J. (1995). Reacting, planning, and learning in an autonomous agent. volume 14, pags. 29–62.

[Benson, 1996] Benson, S. S. (1996). *Learning action models for reactive autonomous agents*. Tesis de doctorado, Stanford University, Stanford, CA, U.S.A.

[Bernard y de la Higuera, 1999] Bernard, M. y de la Higuera, C. (1999). GIFT: Grammatical inference for terms. *International Conference on Inductive Logic Programming*, pags. 1332–1348.

[Billard et al., 2008] Billard, A., Calinon, S., Dillmann, R., y Schaal, S. (2008). Robot programming by demonstration. En Siciliano, B. y Khatib, O., editores, *Handbook of Robotics*. Springer. In press.

[Blockeel y Raedt, 1998] Blockeel, H. y Raedt, L. D. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297.

[Boucher y Dominey, 2006] Boucher, J. y Dominey, P. F. (2006). Perceptual-motor sequence learning via human-robot interaction. *Lecture Notes in Computer Science LNAI 4095*, pags. 224–235.

[Bratko, 1986] Bratko, I. (1986). *Prolog programming for artificial intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Bratko, 1999] Bratko, I. (1999). Refining complete hypotheses in ilp. *ILP '99: Proceedings of the 9th International Workshop on Inductive Logic Programming*, pags. 44–55.

[Bratko, 2000] Bratko, I. (2000). Modelling operator's skill by machine learning. *22nd International Conference Information Technology Interfaces*, pags. 23–30.

[Broda y Hogger, 2004] Broda, K. y Hogger, C. (2004). Designing and simulating individual teleo-reactive agents. En *27th German Conference on Artificial Intelligence, Ulm*, pags. 1–15.

[Byers et al., 2001] Byers, Z., Dixon, M., Smart, W., y Grimm, C. M. (2001). Symbolic dynamic programming for first-order MDPs. volume 14, pags. 690–97.

[Camacho, 1995] Camacho, R. (1995). Using machine learning to extract models of human control skills. *The Second International Workshop on Artificial Intelligence Techniques (AIT-95)*, pags. 143–160.

[Cocora et al., 2006] Cocora, A., Kersting, K., Plagemann, C., Burgard, W., y De Raedt, L. (2006). Learning relational navigation policies. En *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pags. 12–18, Beijing, China.

[de la Higuera, 2000] de la Higuera, C. (2000). Current trends in grammatical inference. En *Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*, pags. 28–31, London, UK. Springer-Verlag.

[de la Higuera y Oats, 2006] de la Higuera, C. y Oats, T. (2006). Grammar induction: Techniques and theory. En *Tutorial held at the 23rd International Conference on Machine Learning (ICML-2006)*, Pittsburgh, Pennsylvania, U.S.A.

[D'Este et al., 2003] D'Este, C., O'Sullivan, M., y Hannah, N. (2003). Behavioural cloning and robot control. En *Robotics and Applications*, pags. 179–182.

[Driessens y Dzeroski, 2004] Driessens, K. y Dzeroski, S. (2004). Integrating guidance into relational reinforcement learning. En *Machine Learning*, pags. 271–304.

[Džeroski et al., 2001] Džeroski, S., De Raedt, L., y Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, 43(1/2):7–52.

[Fikes y Nilsson, 1971] Fikes, R. E. y Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208.

[Franz y Mallot, 2000] Franz, M. y Mallot, H. A. (2000). Biomimetic robot navigation. *Robotics and autonomous Systems*, 30:133–153.

[González et al.,] González, J., Holder, L. B., y Cook, D. J. Experimental comparison of graph-based relational concept learning with inductive logic programming systems.

[Hernández y Morales, 2006] Hernández, S. y Morales, E. (2006). Global localization of mobile robots for indoor environments using natural landmarks. *IEEE International Conference on Robotics, Automation and Mechatronics (RAM)*, pags. 1–6.

[Isaac y Sammut, 2003] Isaac, A. y Sammut, C. (2003). Goal-directed learning to fly. *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, pags. 258–265.

[Jáquez, 2005] Jáquez, V. M. (2005). Construcción de mapas y localización simultánea con robots móviles. Tesis de maestría, Instituto Tecnológico y de Estudios Superiores de Monterrey, Cuernavaca, Morelos, México.

[Kadous et al., 2006] Kadous, M. W., Sammut, C., y Sheh, R. K.-M. (2006). Autonomous traversal of rough terrain using behavioural cloning. En *Proceedings of the 3rd International Conference on Autonomous Robots and Automation*, pags. 219–224, Palmerston North, New Zealand.

[Kaiser y Dillmann, 1996] Kaiser, M. y Dillmann, R. (1996). Building elementary robot skills from human demonstration. En *International Symposium on Intelligent Robotics Systems*, pags. 2700–2705.

[Karalič y Bratko, 1997] Karalič, A. y Bratko, I. (1997). First order regression. volume 26, pags. 147–176, Hingham, MA, USA. Kluwer Academic Publishers.

[Klingspor et al., 1996] Klingspor, V., Morik, K., y Rieger, A. D. (1996). Learning concepts from sensor data of a mobile robot. volume 23, pags. 305–332.

[Klingspor y Sklorz, 1995] Klingspor, V. y Sklorz, S. (1995). Representing, learning, and executing operational concepts. En *Proceedings of the 3rd European Workshop on Learning Robots*, pags. 88–97.

[Kochenderfer, 2003] Kochenderfer, M. J. (2003). Evolving hierarchical and recursive teleo-reactive programs through genetic programming. volume 2610, pags. 84–94.

[Konidaris, 2008] Konidaris, G. (2008). Autonomous robot skill acquisition. En *Twenty-Third AAAI Conference on Artificial Intelligence*.

[Konik y Laird, 2006] Konik, T. y Laird, J. E. (2006). Learning goal hierarchies from structured observations and expert annotations. volume 64, pags. 263–287. Springer Netherlands.

[Lavrac y Dzeroski, 1994] Lavrac, N. y Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York.

[Likhachev y Arkin, 2000] Likhachev, M. y Arkin, R. C. (2000). Robotic comfort zones. volume 4196 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pags. 27–41.

[Mataric, 1998] Mataric, M. J. (1998). Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior. En *Trends in Cognitive Science*, pags. 82–87.

[Michie y Sammut, 1995] Michie, D. y Sammut, C. (1995). Behavioural clones and cognitive skill models. *Machine Intelligence*, 14:395–404.

[Mitchell, 2006] Mitchell, T. (2006). The discipline of machine learning. Technical Report CMU ML-06 108, Carnegie Mellon University.

[Morales, 2006] Morales, E. F. (2006). Programación Lógica Inductiva (ILP). En *Aprendizaje Automático: conceptos básicos y avanzados. Aspectos prácticos utilizando el software WEKA*, pags. 201–221. Basilio Sierra Araujo.

[Morales y Sammut, 2004] Morales, E. F. y Sammut, C. (2004). Learning to fly by combining reinforcement learning with behavioural cloning. En *Proceedings of the twenty-first International Conference on Machine learning*, pags. 598 – 605, Banff, Alberta, Canada.

[Muggleton, 1996] Muggleton, S. (1996). Inverting entailment and Progol. En *Machine intelligence 14: applied machine intelligence*, pags. 133–187, New York, NY, USA. Oxford University Press, Inc.

[Nevill-Manning y Witten, 1997] Nevill-Manning, C. y Witten, I. (1997). Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82.

[Nicolescu y Mataric, 2002] Nicolescu, M. y Mataric, M. J. (2002). A hierarchical architecture for behavior-based robots. *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*.

[Nilsson, 1994] Nilsson, N. J. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, (1):139–158.

[Ogale et al., 2007] Ogale, A. S., Karapurkar, A., y Aloimonos, Y. (2007). View-invariant modeling and recognition of human actions using grammars. *Lecture Notes in Computer Science*, pags. 115–126.

[Pineau et al., 2003] Pineau, J., Montemerlo, M., Pollack, M., Roy, N., y Thrun, S. (2003). Towards robotic assistants in nursing homes: Challenges and results. *Special issue on Socially Interactive Robots, Robotics and Autonomous Systems*, 42:271–281.

[Plotkin, 1969] Plotkin, G. (1969). A note on inductive generalization. *Machine Intelligence*, 5:153–163.

[Pomerleau, 1991] Pomerleau, D. A. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3:97.

[Quinlan, 1993] Quinlan, J. (1993). C4.5: Programs for machine learning. Morgan Kaufmann Publishers.

[Quinlan, 1990] Quinlan, J. R. (1990). Induction of decision trees. En Shavlik, J. W. y Dietterich, T. G., editores, *Readings in Machine Learning*. Morgan Kaufmann. Originally published in *Machine Learning* 1:81–106, 1986.

[Quinlan y Cameron-Jones, 1993] Quinlan, J. R. y Cameron-Jones, R. M. (1993). Foil: A midterm report. En *Proceedings of the European Conference on Machine Learning*, pags. 3–20. Springer-Verlag.

[Russell y Norvig, 2003] Russell, S. y Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition.

[Ryan y Pendrith, 1998] Ryan, M. y Pendrith, M. (1998). RL-TOPs: An architecture for modularity and re-use in reinforcement learning. *Proceedings of The 15th International Conference on Machine Learning*.

[Ryan y Reid, 2000] Ryan, M. y Reid, M. (2000). Learning to fly: An application of hierarchical reinforcement learning. En *Proceedings 17th International Conference on Machine Learning*, pags. 807–814. Morgan Kaufmann.

[Sánchez-Texis, 2008] Sánchez-Texis, J. (2008). Localización local y global integrando información visual invariante y de rango para ambientes no estacionarios. Tesis de maestría, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Tonantzintla, Puebla, México.

[Saunders et al., 2007] Saunders, J., Nehaniv, C. L., Dautenhahn, K., y Alissandrakis, A. (2007). Self-imitation and environmental scaffolding for robot teaching. *International Journal of Advanced Robotics Systems*, 4(1):109–124.

[Sim et al., 2003] Sim, S. K., Ong, K. W., y Seet, G. (2003). A foundation for robot learning. *ICCA '03. Proceedings. 4th International Conference on Control and Automation, 2003.*, pags. 649–653.

[Suc y Bratko, 1997] Suc, D. y Bratko, I. (1997). Skill reconstruction as induction of LQ controllers with subgoals. En *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, volume 2, pags. 914–919.

[Sutton y Barto, 1998] Sutton, R. S. y Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

[Thrun et al., 2000] Thrun, S., Beetz, M., Bennewitz, M., Burgard, W., Cremers, A., Dellaert, F., Fox, D., Hähnel, D., Rosenberg, C., Roy, N., Schulte, J., y Schulz, D. (2000). Probabilistic algorithms and the interactive museum tour-guide robot Minerva. *International Journal of Robotics Research*, 19(11):972–999.

[Tonaru et al., 2009] Tonaru, T., Takiguchi, T., y Ariki, Y. (2009). Extraction of human activities as action sequences using plsa and prefixspan. *International Journal of Hybrid Information Technology*, 2(1):13–20.

[Urbancic y Bratko, 1994] Urbancic, T. y Bratko, I. (1994). Reconstructing human skill with machine learning. *Proceedings of the 11th European Conference on Artificial Intelligence ECAI-94*, pags. 498–502.

[van Zaanen, 2000] van Zaanen, M. V. (2000). ABL: alignment-based learning. En *Proceedings of the 17th Conference on Computational linguistics*, pags. 961–967, Morristown, NJ, USA. Association for Computational Linguistics.

[Vaughan et al., 2003] Vaughan, R. T., Gerkey, B. P., y Howard, A. (2003). On device abstractions for portable, reusable robot code. En *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pags. 2421–2427.

[Vijaykumar et al., 1987] Vijaykumar, R., Venkataraman, S., Dakin, G., y Lyons, D. M. (1987). A task grammar approach to the structure and analysis of robot programs. Technical Report UM-CS-1987-067, Amherst, MA, USA.

[Westeyn et al., 2003] Westeyn, T., Brashear, H., Atrash, A., y Starner, T. (2003). Georgia Tech gesture toolkit: supporting experiments in gesture recognition. En *Proceedings of the 5th international Conference on Multimodal Interfaces ICMI-03*, pags. 85–92.

[Zelek y Levine, 1996] Zelek, J. S. y Levine, M. D. (1996). SPOTT: A mobile robot control architecture for unknown or partially known environments. En *AAAI Spring Symposium on*, pags. 25–27. AAAI Press.

Referencias en línea

[Blackburn y Striegnitz, 2002] Blackburn, P. y Striegnitz, K. (2002). Natural language processing techniques in Prolog-url. http://cs.union.edu/~striegnk/courses/nlp-with-prolog/html/index.html, Fecha de consulta: 20 de Octubre de 2008.

[Champandard, 2007] Champandard, A. J. (2007). AI-Game-Dev: Artificial Intelligence Game Development. http://aigamedev.com/theory/teleo-reactive-programs-agent-control.

[ifr url, 2008] ifr url (2008). International federation of robotics. http://www.ifr.org/. Fecha de consulta: 2 de Junio de 2008.

[iRobot url, 2008] iRobot url (2008). iRobot: Roomba family. http://www.irobot.com/. Fecha de consulta: 4 de Junio de 2008.

[Kochenderfer,] Kochenderfer, M. J. TRSoccerbots-url. http://www.trsoccerbots.org/. Fecha de consulta: 11 de Abril de 2006.

[Robertson, 1998] Robertson, D. S. (1998). Quickprolog online book-url. http://www.dai.ed.ac.uk/groups/ssp/bookpages/quickprolog/quickprolog.html. Fecha de consulta: 14 de Julio de 2008.

[Robocup url, 2008] Robocup url (2008). Robocup at home. http://www.ai.rug.nl/robocupathome/. Fecha de consulta: 26 de Agosto de 2008.

[Robotics, 2006] Robotics, A. (2006). http://www.activmedia.com/. Fecha de consulta: 5 de Marzo de 2006.

[Skilligent-url, 2008] Skilligent-url (2008). Task and skill learning for multifunctional service robots. http://www.skilligent.com/. Fecha de consulta: 23 de Septiembre de 2008.

[Srinivasan-ALEPH-url, 1999] Srinivasan-ALEPH-url, A. (1999). The ALEPH 5 manual. http://web2.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html. Fecha de consulta: 4 de Marzo de 2006.

[TMR07buscar-url, 2007] TMR07buscar-url (2007). Torneo mexicano de robótica 2007: Markovito buscando un objeto. http://www.youtube.com/watch?v=H5mPHgyzexc. Fecha de consulta: 8 de Noviembre de 2007.

[TMR08nab-url, 2007] TMR08nab-url (2007). Torneo mexicano de robótica 2008: Markovito navegando. http://www.youtube.com/watch?v=R45e97WpmH4. Fecha de consulta: 20 de Octubre de 2007.

[TMR08seguir-url, 2008] TMR08seguir-url (2008). Torneo mexicano de robótica 2008: Markovito siguiendo a una persona. http://www.youtube.com/watch?v=AM8MxpW1kx0. Fecha de consulta: 2 de Septiembre de 2008.

Publicaciones derivadas de la tesis

[Avilés et al., 2007] Avilés, H. H., Corona, E., Ramírez, A., Vargas, B., Sánchez, J., Sucar, L. E., y Morales, E. F. (2007). A service robot named Markovito. *4th IEEE LARS 07 / IX COMRob 07.*

[Avilés et al., 2009] Avilés, H. H., Sucar, L. E., Morales, E. F., Vargas, B. A., Sánchez, J., y Corona, E. (2009). Markovito: A flexible and general service robot. En *Studies in Computational Intelligence*, volume 177, pags. 401–423. Springer Berlin / Heidelberg.

[Vargas y Morales, 2008] Vargas, B. y Morales, E. F. (2008). Solving navigation tasks with learned teleo-reactive programs. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Poster.*

[Vargas y Morales, 2009] Vargas, B. y Morales, E. F. (2009). Learning navigation teleo-reactive programs using behavioural cloning. *IEEE International Conference on Mechatronics (ICM).*

[Vargas-Govea y Morales, 2009] Vargas-Govea, B. y Morales, E. F. (2009). Learning relational grammars from sequences of actions. *14th Iberoamerican Congress on Pattern Recognition.*

A

Conocimiento del dominio

El conocimiento del dominio es información adicional que el usuario puede proporcionar directamente o que también puede aprenderse. A continuación se describen los predicados correspondientes al conocimiento del dominio de los PTRs de navegación inducidos.

A.1 *Deambular*

$mayorig\_que(+X, +Valor)$	Es verdadero si X es menor o igual que Valor
$menor\_que(+X, +Valor)$	Es verdadero si X es menor que Valor
$distancia(+Marca, -Distancia)$	Obtiene la Distancia relativa de la marca al robot
$angulo(+Marca, -Angulo)$	Obtiene el Angulo de la marca
$obten\_atributo(+Marca, -Atributo)$	Obtiene el Atributo de la marca
$ltipo(marca(\_, \_, \_, Ltipo), Ltipo)$	Obtiene el tipo de la marca: `pared`, `esquina` o `discontinuidad`
$min\_busca([], +Lista, -Marca)$	Busca en Lista la Marca más cercana
$calcula\_Thg(+Xr, +Yr, +Xg, +Yg, -Thg)$	Obtiene la orientación hacia la meta Thg con base en la posición del robot Xr,Yr y la de la meta Xg,Yg

A.2 *Salir de trampa*

$veo\_pared(+Estado, -Pared)$	Identifica en el Estado si hay una Pared. Los valores de Pared pueden ser: `paredizquierda`, `paredderecha` o `paredfrontal`

A.3 *Seguir*

En el aprendizaje del PTR para ***seguir*** se utiliza el mismo conocimiento del dominio que para aprender el PTR para ***deambular***.

B

Conceptos y PTRs aprendidos

A continuación se muestran diversos conceptos y PTRs aprendidos para las tareas de navegación.

B.1 *Deambular*

Concepto:
zona_frontal(+Estado,-Valor)
lee del Estado el Valor correspondiente a la `zona_frontal`. Los valores pueden ser: `conObstaculo` y `sinObstaculo`.

```
zona_frontal(Estado,conObstaculo) :-
    obst_mas_cercano(Estado,_,Distancia,Angulo),
    menor_que(Distancia,0.49),
    Angulo =\= 90,
    Angulo =\= -90.
zona_frontal(Estado,sinObstaculo) :-
    obst_mas_cercano(Estado,_,Distancia,Angulo),
    Distancia > 0.49.
zona_frontal(Estado,sinObstaculo) :-
    obst_mas_cercano(Estado,_,Distancia,Angulo),
    menor_que(Distancia,0.49),
    Angulo == 90.0.
zona_frontal(Estado,sinObstaculo) :-
    obst_mas_cercano(Estado,_,Distancia,Angulo),
    menor_que(Distancia,0.49),
    Angulo == -90.0.
```

PTR:
deambular(+Estado, -Acción)
De acuerdo a la información dada por el Estado, devuelve la Acción que el robot debe ejecutar para deambular. Las acciones pueden ser `girarIzquierda`, `girarDerecha` y `avanzar`.

```
deambular(Estado, girarIzquierda) :-
    zona_frontal(Estado,conObstaculo),
    obst_mas_cercano(Estado, _, _, Angulo),
    menor_que(Angulo, -1.51).
deambular(Estado, girarDerecha) :-
    zona_frontal(Estado,conObstaculo).
deambular(Estado, avanzar) :-
    zona_frontal(Estado,sinObstaculo).
```

B.2 *Orientarse*

PTR:
orientar(+Estado,-Accion)
De acuerdo a la información dada por el Estado, devuelve la Acción que el robot debe ejecutar para orientarse. Las acciones pueden ser `girarIzquierda` y `girarDerecha`.

```
orientar(Estado,Accion) :-
    zona_frontal(Estado,sinObstaculo),
    deambular(Estado,Accion).
orientar(Estado,girarIzquierda) :-
    zona_orienta(Estado,giroSeguro),
    obten_orientacion(Estado, Thr, Thg),
    headings(Thr, Thg, diferente),
    closer(Thg,Thr).
orientar(Estado, Accion) :-
    zona_orienta(Estado,giroSeguro),
    get_pose(Estado, Thr, Thg),
    obten_orientacion(Thr, Thg, igual),
    deambular(Estado,Accion).
orientar(Estado,giroDerecha) :-
    zona_orienta(Estado,giroSeguro).
orientar(Estado,Accion) :-
```

```
    zona_orienta(Estado,noreconocido),
    deambular(Estado,Accion).
```

B.3 *Salir de trampa*

Concepto:
config_paredes(+ParedDer,+ParedIzq,+ParedFrente,-Valor)
De las distancias: `ParedDer`, `ParedIzq` y `ParedFrente` calcula Valor: si el robot está dentro de una trampa (`trapin`), está fuera de ella (`trapout`) o está saliendo (`encurso`).

```
config_paredes(ParedDer,ParedIzq,ParedFrente,trapin) :-
    wdistancia(ParedDer,riesgosa),
    wdistancia(ParedIzq,riesgosa).
config_paredes(ParedDer,ParedIzq,ParedFrente,trapout) :-
    wdistancia(ParedDer,segura),
    wdistancia(ParedIzq,segura),
    wdistancia(ParedFrente,segura).
config_paredes(_,_,_,encurso).
```

Concepto:
wlado(+ParedDer,+ParedIzq,-Libre)
De las distancias: `ParedDer` y `ParedIzq` y `ParedFrente` determina el lado hacia donde el robot está libre (`derecha-libre`/`izquierda-libre`) o si está atrapado (`atrapado`).

```
wlado(ParedDer,ParedIzq,derecha-libre) :-
    distancia(ParedDer,Rdist),
    angulo(ParedDer,Rangulo),
    distancia(ParedIzq,Ldist),
    angulo(ParedIzq,Langulo),
    mayorig_que(Rdist,0.7),
    mayorig_que(Rdist,Ldist),
    menor_que(Rangulo, -33.8),
    menor_que(Rangulo, 0).
wlado(ParedDer,ParedIzq,izquierda-libre) :-
    distancia(ParedDer,Rdist),
    angulo(ParedDer,Rangulo),
```

```
    distancia(ParedIzq,Ldist),
    angulo(ParedIzq,Langulo),
    mayorig_que(Ldist,0.7),
    mayorig_que(Ldist,Rdist),
    mayorig_que(Langulo, 29.78),
    menor_que(Langulo, 90).
wlado(ParedDer,ParedIzq,atrapado) :-
    distancia(ParedDer,Rdist),
    angulo(ParedDer,Rangulo),
    distancia(ParedIzq,Ldist),
    angulo(ParedIzq,Langulo).
```

Concepto:
wdistancia(+Pared,+Valor)
Determina a partir de la distancia Pared el Valor: `riesgosa` o `segura`.

```
wdistancia(pared,riesgosa) :-
    distancia(pared,D),
    menor_que(D,0.7).
wdistancia(pared,segura) :-
    distancia(pared,D),
    mayorig_que(D,0.7).
```

Concepto:
deja_trampa(+Estado, -Accion)
Obtiene del Estado la Acción que el robot tiene que ejecutar para poder salir de una trampa. Las posibles acciones son: `retrocede`, `giroDerecha` y `giroIzquierda`.

```
deja_trampa(Estado, retrocede) :-
    veo\_pared(Estado, Pared),
    config_paredes(Pared,ParedDer,ParedIzq,ParedFrente,trapin).
deja_trampa(Estado, nada) :-
    veo\_pared(Estado, Pared),
    config_paredes(Pared,ParedDer,ParedIzq,ParedFrente,trapout),
    zona_frontal(Estado, libre).
deja_trampa(Estado, giroDerecha) :-
    veo\_pared(Estado, Pared),
    config_paredes(Pared,ParedDer,ParedIzq,ParedFrente,enproceso),
```

```
    wlado(ParedDer, ParedIzq, derecha-libre).
deja_trampa(Estado, giroIzquierda) :-
    veo\_pared(Estado, Pared),
    config_paredes(Pared,ParedDer,ParedIzq,ParedFrente,enproceso),
    wlado(ParedDer, ParedIzq, izquierda-libre).
```

B.4 *Seguir*

Concepto:
rango_distancia(+Objeto, -Rango)
Determina si la distancia al Objeto está en alguno de los siguientes valores de Rango: `riesgoso`, `lejos`, `menoslejos` o `cerca`.

```
rango_distancia(A, riesgosa) :-
    menor_que(A, 0.28).
rango_distancia(A, lejos) :-
    menor_que(A, 1.0),
    mayorig_que(A, 0.6).
rango_distancia(A, menoslejos) :-
    menor_que(A, 0.6),
    mayorig_que(A, 0.40).
rango_distancia(A, cerca) :-
    menor_que(A, 0.40),
    mayorig_que(A, 0.28).
```

Concepto:
zona_seguir(+Estado, -Valor)
Determina a partir del Estado el rango de la zona de seguimiento y si existe un objeto para que el robot pueda seguir. Valor puede tomar los valores: `seguir` ó `noseguir`.

```
zona_seguir(R, seguir):-
    R < 137.77,
    R > 49.27.
zona_seguir(R, noseguir) :-
    mayorig_que(R, 137.77).
```

```
zona_seguir(R, noseguir) :-
    menor_que(R, 49.27).
```

PTR:
sigue(+Marca, -Distancia, -Angulo, -Accion)
Determina a partir de la Marca si el objeto está en la zona de seguimiento y la Acción a seguir. Las posibles acciones son: `retrocede`, `giroDerecha`, `giroIzquierda`, `avanzarapido`, `avanzamedio` y `avanzalento`. Si no existe una Marca en la zona de seguimiento entonces el robot emite un mensaje.

```
sigue(Marca, Distancia, Angulo, retrocede) :-
    rango_distancia(Distancia, riesgosa),!.
sigue(Marca, Distancia, Angulo, nada) :-
    zona_seguir(Angulo,seguir),
    rango_distancia(Distancia, cerca),!.
sigue(Marca, Distancia, Angulo, giroDerecha) :-
    zona_seguir(Angulo,seguir),
    menor_que(Angulo, 70.39).
sigue(Marca, Distancia, Angulo, giroIzquierda) :-
    zona_seguir(Angulo,seguir),
    mayorig_que(Angulo, 105.59).
sigue(Marca, Distancia, Angulo, avanzarapido) :-
    zona_seguir(Angulo,seguir),
    rango_distancia(Distancia, far1),!.
sigue(Marca, Distancia, Angulo, avanzamedio) :-
    zona_seguir(Angulo,seguir),
    rango_distancia(Distancia, avanzalento),!.
sigue(Estado,_,_,mensaje). %handmade
```

B.5 *Ir a un punto*

PTR:
ir_a(+Estado, -Accion)
Determina a partir del Estado la Acción a ejecutar para ir a un punto. Las posibles acciones son: `avanzar`, `girarIzquierda` y `girarDerecha`.

```
ir_a(Estado,nada,_,_) :-
```

```
    en_meta(Estado,nil).
ir_a(Estado,Accion,_,_) :-
    orientar(Estado,nada),!,
    deambular(Estado,Accion),!.
ir_a(Estado,Accion,_,_) :-
    deambular(Estado,avanzar),!,
    orientar(Estado,Accion),!.
```