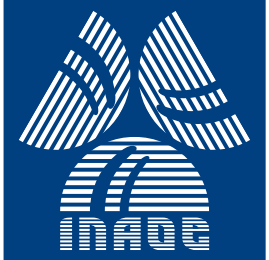


Aprendizaje por Refuerzo Relacional con Acciones Continuas

por

Julio César Hernández Zaragoza

Tesis sometida como requisito parcial para obtener el grado de

Maestro en Ciencias en el Área de Ciencias Computacionales en el

Instituto Nacional de Astrofísica, Óptica y Electrónica

Supervisada por:

Dr. Eduardo F. Morales Manzanares, INAOE

Resumen

El Aprendizaje por Refuerzo (*Reinforcement Learning, RL*) es una técnica muy utilizada para el aprendizaje de tareas en robótica. Esto se debe, principalmente, a que permite a los agentes o robots generar políticas de control a través de interacciones de prueba y error con el ambiente en el cual se encuentran estos robots y a que no se requiere un modelo previo de dicho ambiente. Sin embargo, los algoritmos tradicionales de *RL* requieren tiempos de entrenamiento muy largos los cuales pueden llegar a ser de varias horas, no son capaces de re-utilizar las políticas aprendidas en dominios o tareas similares y ejecutan acciones discretas.

En espacios de búsqueda muy grandes con miles de posibles estados, el proceso de generación de la política puede consumir algunas horas y aunado a esto, una vez generada la política, si la meta de la tarea o el ambiente cambian, es necesario generar una nueva política que tome en cuenta tales cambios. Finalmente, las acciones discretas producen movimientos o desplazamientos poco precisos por parte del robot el cuál puede llegar a acumular errores de hasta decenas de grados para acciones de giro y de hasta decenas de centímetros para el caso de acciones de desplazamiento. Además, las acciones discretas producen trayectorias más lentas que las acciones continuas ya que, con acciones discretas, el robot necesita detenerse para posteriormente girar en ángulos discretos incrementando así, cada vez que se detiene, el tiempo de ejecución de sus tareas.

En este trabajo se presenta un método en dos fases para abordar estos problemas. En

la primera fase, la información de bajo nivel de los sensores del robot se transforma en una representación relacional de estados y acciones basada en habitaciones, pasillos, puertas, paredes y obstáculos con la cual reduce significativamente el espacio de estados. Se empleó Clonación de Comportamiento (*Behavioural Cloning, BC*), es decir, trazas proporcionadas por el usuario para aprender, en pocas iteraciones, una política de control la cual, debido a las descripciones relacionales, puede ser re-utilizada en dominios o ambientes diferentes. Sin embargo, esta política hace uso de acciones discretas. En la segunda fase, se utiliza una Regresión Pesada Local (*Locally Weighted Regression, LWR*) para transformar la política con acciones discretas en una política con acciones continuas.

El método fue utilizado para generar políticas de control de navegación y de seguimiento para robots móviles simulados y reales con resultados muy prometedores. Los resultados muestran cómo las políticas son aprendidas en pocas iteraciones, pueden ser utilizadas en diferentes dominios, ejecutan trayectorias más suaves, más cortas y en menos tiempo que las políticas relacionales originales y la calidad de las tareas se aproxima a la calidad de ejecución de las trazas proporcionadas por el usuario.

Abstract

Reinforcement Learning (RL) is a commonly used technique for learning tasks in robotics. This is mainly because it allows agents, i.e., robots, to develop optimal control policies through trial and error interactions with the environment in which these robots perform and because it does not require a previous model of such environment. However traditional RL algorithms require long training times which can be several hours, are unnable to re-use learned policies in similar domains or similar tasks and perform discrete actions.

In large search spaces with thousands of states, the policy generation process takes some hours and besides, once a policy has been generated, if the goal or the environment changes, a new policy has to be generated in order to take into account such changes. Finally, discrete actions produce imprecise movements by the robot which can accumulate an error up to tens of degrees for turning actions and up to tens of centimeters for displacement actions. Besides, discrete actions produce slower paths than continuous actions since, with discrete actions, the robot needs to stop in order to turn in discrete angles increasing, every time it stops, the tasks' execution times.

In this work, a two stage method to tackle these problems is presented. In the first stage, the low level sensor information coming from the robot's sensors is transformed into a relational description based on rooms, corridors, doors, walls and obstacles to characterize states and actions, significantly reducing the state space. Behavioural Cloning (BC), i.e., traces provided by the user, are used to learn in few iterations, a control policy, which, due to

the relational representation, can be re-used in similar but different domains or environments. However, this policy uses discrete actions. In the second stage, Locally Weighted Regression (LWR) is used to transform the discrete actions policy into a continuous actions policy.

The method was used to generate control policies for navigation and following tasks for simulated and real mobile robots with very promising results. The results show that the policies are learned after few iterations, can be used on different domains, perform smoother, faster and shorter paths than the original relational policies and the tasks' quality is similar to the traces provided by the user.

Contenido

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

1 Introducción

1.1 Motivación

Un robot de servicio es un robot que opera de forma parcial o totalmente autónoma para realizar servicios útiles para el bienestar de los humanos[1]. En la actualidad es posible encontrar robots de servicio para operaciones de limpieza, mantenimiento, asistencia, ayuda, cuidados, seguridad, transporte, exploración, medicina, educación, entretenimiento, etc. Debido a la variedad de servicios que proporcionan, en los últimos años se ha observado un incremento en el uso de los robots de servicio en hogares y centros de trabajo y se espera que este incremento sea aún mayor en los próximos años[2]. Sin embargo, la completa incorporación, aceptación y uso de los robots de servicio dependerá, en gran medida, de su capacidad para aprender nuevas tareas.

Programar robots para que aprendan nuevas tareas puede convertirse en un proceso que consume grandes cantidades de tiempo. Además de que la forma en la cual se tiene que especificar cómo hacer el mapeo de la información de los sensores a los activadores del robot

[1]Definición según la Federación Internacional de Robótica (*International Federation of Robotics*, *IFR*).

[2]Según estadísticas del *IFR Statistical Department*, publicadas en el *IFR World Robotics Report*, 2008.

es propensa a errores de programación y a largas jornadas de depuración. La idea de hacer que un robot aprenda por sí solo cómo realizar una tarea en lugar de indicarle, de forma explícita, cómo hacerlo, es mucho más atractiva. Además, es más fácil y mucho más intuitivo especificar lo que el robot tiene que hacer y dejarlo que aprenda por sí solo los detalles finos de cómo hacerlo.

Una forma en la cual el robot puede aprender a realizar nuevas tareas, es a través de la interacción directa con su ambiente. El aprendizaje por refuerzo ha sido ampliamente utilizado y sugerido como buen candidato para el aprendizaje de tareas en robótica: (Schaal y Atkeson 1991, Lin 1993, Mahadevan y Connell 1991, Smart y Kaelbling 2002, Smart 2002, Peters et al. 2003, Wang et al. 2003, Morales 2005, Lee. et al. 2006, Conn y Peters 2007, Cuaya y Munoz 2007, Torrey et al. 2008). Esto se debe, principalmente, a que permite a un agente, es decir al robot, generar políticas de control mientras interactúa con su ambiente y a que no se requiere un modelo previo de tal ambiente. En aprendizaje por refuerzo, básicamente se le indica al robot qué es lo que debe de hacer y éste, de forma "autónoma" a través de prueba y error, aprende a desarrollar comportamientos que le permitan completar la tarea.

1.2 Problemática

El uso y aplicación de técnicas tradicionales de aprendizaje por refuerzo, e.g., (Watkins 1989, Schaal y Atkeson 1991, Mataric 1994, Sutton y Barto 1998, Dzeroski et al. 1998), se ha visto limitado debido a cuatro aspectos principales:

1. Cantidad de datos generados por los sensores del robot.

2. Tamaño de los espacios de estados.

3. Inhabilidad para transferir o re-utilizar políticas previamente aprendidas.

4. Acciones discretas.

Los robots normalmente están equipados con distintos grupos de sensores, entre ellos se pueden encontrar: sensores láser de rango, sensores de contacto, anillos de sonares, cámaras, etc. Todos estos sensores producen una enorme cantidad de lecturas a tasas muy altas de muestreo. Esto ocasiona problemas a muchos de los algoritmos de aprendizaje existentes debido a la gran cantidad de información que se tiene que procesar.

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 los espacios de estados muy grandes, del orden de cientos de miles de estados, por lo general se requiere de un gran número de iteraciones o de tiempos de entrenamiento muy largos antes de poder desarrollar una política de control aceptable. Esto es un problema para la robótica móvil y de servicio, donde el espacio de estados normalmente es continuo, es decir, las variables involucradas están definidas por valores reales por lo que es difícil aplicar algoritmos clásicos de aprendizaje debido a la cantidad de información de bajo nivel que se adquiere con el sensado continuo; además de que la descripción de los estados, por lo general, involucra una gran cantidad de variables. Para solucionar este problema se han propuesto diferentes estrategias, las cuales generalmente se basan en una discretización del espacio de estados. Sin embargo, las políticas aprendidas de esta forma sólo son aplicables en el ambiente en el cual fueron generadas y no pueden ser re-utilizadas en algún otro dominio similar. A este tipo de políticas que no pueden ser tranferibles o re-utilizadas se les conoce como **políticas proposicionales**. En robótica, lo que se busca es que el robot aprenda, por ejemplo, cómo salir de una habitación y que después sea capaz de salir de cualquier habitación con características similares, sin la necesidad de volver a generar una nueva política o de volver a entrenarlo desde el principio. Esto es, se busca que todo o parte de la política de control que generó el robot al desempeñar una tarea específica, pueda ser transferido o re-utilizado cuando se presenta una situación nueva o desconocida en una tarea similar. Al tipo de políticas que pueden ser tranferibles o re-utilizadas se les conoce como **políticas relacionales** (Dzeroski et al. 1998, Morales 2005,

Cocora et al. 2006, Torrey et al. 2008, Walker et al. 2008).

En muchos enfoques de aprendizaje se asumen acciones discretas. A pesar de ser útiles en otros dominios, en robótica la ejecución de acciones discretas produce movimientos poco naturales o bruscos por parte del robot, los tiempos de ejecución de las tareas son más extensos, las trayectorias son más largas y propician la acumulación de una mayor cantidad de errores odométricos, al ser comparadas contra acciones discretas.

En el presente trabajo de tesis se emplean las ventajas, como son simplicidad, autonomía y no dependencia del modelo, que ofrece aprendizaje por refuerzo (en particular *Q-learning*), para el desarrollo de un método de generación de políticas de control que puedan ser transferibles o re-utilizables, que permitan la ejecución de acciones continuas y que puedan ser empleadas para el aprendizaje de tareas con robots móviles y de servicio. Para ello se desarrolló un método en dos fases llamado *TS-RRLCA* (*Two-Stage Relational Reinforcement Learning with Continuous Actions*). En la primera fase, las lecturas de bajo nivel de los sensores del robot son transformadas en conjuntos de información más reducidos y descriptivos. Estos conjuntos constan de marcas naturales del ambiente y se utilizan para generar representaciones relacionales las cuales describen los estados del ambiente y las acciones del robot. Se utiliza clonación de comportamiento, es decir, trazas para obtener conjuntos de pares estado-acción relevantes para el aprendizaje de las tareas, disminuyendo así el espacio de búsqueda. Estos conjuntos de pares estado-acción se proporcionan a un algoritmo de aprendizaje por refuerzo el cual genera, en pocas iteraciones, una política de control relacional con acciones discretas.

En la segunda fase del método, la política relacional con acciones discretas generada anteriormente, es transformada en una política relacional con acciones continuas a traves de regresiones pesadas locales. Dado que la regresión es local a un punto en particular, esta segunda fase también se realiza de forma rápida, permitiendo así utilizar el método para la generación de políticas de control que pueden ser utilizadas con robots móviles de servicio reales.

1.3 Objetivo de la tesis

Desarrollar un método para aprender políticas de control relacionales, con acciones continuas para tareas de navegación y de seguimiento para robots móviles.

Los objetivos particulares del presente trabajo de tesis son:

- Desarrollar un método para disminuir la cantidad de datos provenientes de los sensores del robot.
- Hacer uso de representaciones relacionales que describan de forma más comprensible para los usuarios, i.e., en un lenguaje más cercano al lenguaje natural, los estados del ambiente y las acciones del robot.
- Desarrollar políticas de control re-utilizables o transferibles aplicando aprendizaje por refuerzo a los estados y acciones relacionales.
- Acelerar el proceso de generación de políticas de aprendizaje por refuerzo a través de clonación de comportamiento.
- Transformar las políticas relacionales con acciones discretas a políticas relacionales con acciones continuas, a través de regresiones pesadas locales.
- Probar las políticas generadas en un robot móvil simulado y en un robot real.

1.4 Contribuciones

Las principales contribuciones del presente trabajo de tesis son las siguientes:

- Representación de alto nivel para caracterizar estados y acciones del robot. 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 utiliza para generar una representación relacional basada en puertas, paredes, esquinas, pasillos, habitaciones y obstáculos que facilita el uso de algoritmos de aprendizaje y que permite transferir conocimiento entre dominios.

- Algoritmo de aprendizaje de estados y acciones relacionales utilizando clonación de comportamiento y predicados de primer orden. A partir de ejemplos o trazas generadas por el usuario se generan secuencias de pares estado-acción que, haciendo uso de nuestra representación de alto nivel, describen los sitios que visitó el robot al realizar su tarea y la correspondiente maniobra que se realizó en cada uno de estos sitios.

- Algoritmo de aprendizaje por refuerzo que genera políticas de control a partir de conjuntos de pares estado-acción relacionales. Este algoritmo toma como entrada un conjunto de pares estado-acción relacionales y a partir de este conjunto genera una política de control relacional con acciones discretas.

- Algoritmo de transformación de políticas relacionales discretas a continuas empleando clonación de comportamiento y regresión pesada local. Este algoritmo utiliza la información de las trazas que generan los usuarios para, a través de un esquema de regresión pesada local, transformar políticas de control relacionales con acciones discretas en políticas relacionales con acciones continuas.

1.5 Alcances

Los alcances de la presente tesis son los siguientes:

- Se generarán dos políticas: una política para que el robot aprenda a desplazarse de un lugar a otro dentro de su ambiente, a la cual se le denominará **política de navegación**,

y una política para que el robot aprenda a seguir a otro robot, denominada **política de seguimiento**.

- Dada la naturaleza, i.e., las dos fases del método, las dos políticas serán generadas en dos versiones, una versión que hace uso de acciones discretas y otra versión que hace uso de acciones continuas.

- Las dos políticas, en ambas versiones, serán generadas en el simulador *player/stage* en un ambiente de simulación determinado.

- Para evaluar que las políticas generadas con este método son transferibles o re-utilizables, serán probadas en un robot simulado el cual es un *Pioneer2DX* de *ActivMedia* en diferentes ambientes simulados, y también serán probadas en un robot real el cual es un *GuiaBot* de *ActivMedia* en un ambiente real.

- Los sensores de los robots (tanto el simulado, como el real) serán láser y sonares.

- Se evaluará el tiempo de generación de las políticas de control y la calidad de ejecución de las tareas (tiempo de ejecución, error respecto a esas mismas tareas cuando son ejecutadas por humanos y porcentaje de celdas de ocupación libres (Romero et al. 2001) que se utilizaron al desempeñar las tareas).

1.6 Organización de la tesis

El presente trabajo de tesis se encuentra organizado de la siguiente manera. En el Capítulo 2 se expone el Marco Teórico, este presenta las bases, algoritmos, métodos y conceptos que dan soporte al desarrollo de la tesis. En el Capítulo 3 se presenta el Trabajo Relacionado sobre los desarrollos de métodos, técnicas y algoritmos que permiten generar políticas o esquemas de control relacionales, que puedan ser aplicados a grandes espacios de búsqueda o en robots reales y que permitan la ejecución de acciones continuas. El Capítulo 4 expone

a detalle el método en dos fases propuesto en el presente trabajo de tesis. El Capítulo 5 presenta los experimentos realizados y los resultados obtenidos de dichos experimentos. El Capítulo 6 puntualiza conclusiones y aportaciones del presente trabajo de tesis y proporciona direcciones de trabajo para futuros desarrollos.

2

Marco Teórico

En el presente capítulo se exponen los principales métodos, algoritmos y técnicas que se utilizan como base para el entendimiento de los conceptos que se emplean en la tesis. En particular se describe el funcionamiento de *Q-learning* uno de los algoritmos más utilizados de aprendizaje por refuerzo el cual es la base del método propuesto. Con *Q-learning* es posible generar políticas de control para diversas tareas en robótica, sin embargo estas políticas no son transferibles y hacen uso de acciones discretas, el objetivo es convertir tales políticas a políticas transferibes con acciones continuas y además acelerar el proceso de generación de estas políticas.

2.1 Aprendizaje por refuerzo

El objetivo general del aprendizaje computacional es el desarrollo de componentes de hardware-software (agentes) inteligentes a través de procesos de aprendizaje y evolución. Aprendizaje por refuerzo es uno de estos procesos.

El aprendizaje por refuerzo consiste en "aprender a asociar situaciones con acciones de modo que se maximice una señal numérica de refuerzo mediante la experimenta-

ción" (Sutton y Barto 1998).

En aprendizaje por refuerzo un agente aprende a través de interacciones con un ambiente dinámico y en ocasiones desconocido. Este agente está conectado al entorno a través de sus sensores y activadores. Con ellos es capaz de ejecutar acciones y de percibir los resultados de tales acciones. A partir de estos resultados el agente modifica su comportamiento. Al ejecutar acciones y observar el resultado o recompensas de tales acciones, el esquema de selección de acciones se va refinando de forma gradual. Eventualmente, si se observa un número suficiente de estados, se genera una política óptima de toma de decisiones y se obtiene un agente que se desempeña con base en esa política en ese ambiente en particular.

Este esquema es adecuado cuando no existe un conocimiento *a priori* del entorno, cuando generar el conjunto de ejemplos para entrenar o aprender a solucionar la tarea es muy complicado, cuando existe incertidumbre sobre las consecuencias de las acciones del agente o en problemas que involucran un conjunto de decisiones secuenciales cuyo resultado o utilidad se conoce hasta el final. Algunos ejemplos son:

- Robótica móvil: aprendizaje de secuencias de movimiento para robots con múltiples extremidades, aprendizaje de la forma de escapar de un laberinto, aprendizaje de la política o secuencia de acciones necesarias para ir de un punto o lugar de origen a un punto de destino dentro de un ambiente determinado.

- Juegos de ajedrez: aprendizaje de la mejor secuencia de movimientos para ganar un juego.

- Manipulación de brazos mecánicos: aprendizaje de la secuencia de pares a aplicar a las articulaciones para conseguir cierto movimiento, aprendizaje de secuencias para tomar y manipular un objeto.

Este esquema de aprendizaje (causa y efecto) es una de las formas básicas de generación de conocimiento que la mayoría de los seres vivos desarrollan a lo largo de sus vidas.

Formalmente, el modelo clásico de aprendizaje por refuerzo consta de:

- S: conjunto finito de posibles estados en los que se puede encontrar el agente, los cuales son de un mismo tamaño fijo previamente definido.
- A: conjunto finito de acciones que puede ejecutar el agente, definidas de forma previa, cuyo valor no puede cambiar, por ejemplo, avanzar 1 m., girar a la derecha -90°, girar a la izquierda 90°, retroceder -1 m., etc.
- r: conjunto de valores escalares de recompensa que indican lo deseable de cada estado, típicamente [0,100].

En aprendizaje por refuerzo el agente está conectado a su entorno vía percepción y acción. En cada paso de interacción o de tiempo (t) el agente percibe a través de sus sensores, el estado ($s_t \in S$) en el que se encuentra. En su estado actual el agente elige una acción $a_t \in A$ y la ejecuta. Tal acción lleva al agente a un nuevo estado $s_{t+1} \in S$. Sin embargo, debido a que el agente no conoce el mundo o ambiente en el que está inmerso, no sabe de antemano, es decir, antes de ejecutar la acción, si dicha maniobra o movimiento es útil para completar la tarea. Se da cuenta si fue útil o no hasta que llega al nuevo estado ya que en este nuevo estado el agente recibe una señal o valor de refuerzo $r_t \in r$. Este valor informa al agente de la utilidad de ejecutar la acción a_t, en el estado s_t para lograr un objetivo concreto. El trabajo del agente consiste entonces en encontrar una política que le permita determinar, ante una situación o estado particular, qué acción es la más adecuada para lograr un objetivo o completar una tarea.

2.1.1 Q-learning

Uno de los desarrollos más importantes de los métodos de aprendizaje por refuerzo, fue la creación de un algoritmo conocido como *Q-learning*. En *Q-Learning* el agente actualiza

una función de utilidad $Q(s_t, a_t)$. Esta función $Q(s_t, a_t)$ representa la utilidad que obtiene al ejecutar la acción a_t en el estado s_t, es decir, esta función es la política de ejecución de acciones que debe utilizar el agente para completar su tarea. En *Q-learning* a esta función de utilidad se le llama función Q.

La idea principal es actualizar la función de utilidad $Q(s_t, a_t)$ de la siguiente forma (Watkins 1989):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.1}$$

Conocido $Q(s_t, a_t)$, es decir, conocida la política el sistema sabe qué acción tiene que seleccionar y ejecutar en cada estado. La acción a ejecutar en cada estado es la que, por lo general, ofrece el valor máximo $Q(s_t, a_t)$.

Exploración vs. explotación

Cuando un ambiente o mundo es desconocido surgen dos objetivos para la selección de acciones que se contraponen:

- El primero es obtener altas recompensas a corto plazo mediante la selección de acciones que lleven a estados buenos ya conocidos (explotar).
- El segundo es aprender a relacionar el estado, la acción, y la recompensa seleccionando acciones no antes probadas o acciones que conduzcan a estados desconocidos (explorar).

Para alcanzar un balance entre exploración y explotación existen tres esquemas principales de selección de acciones:

- *greedy*: en donde siempre se selecciona la acción que da el mayor valor estimado o valor Q.

- ϵ-*greedy*: en donde la mayor parte del tiempo, con cierta probabilidad P se selecciona la acción que da el mayor valor Q, pero con probabilidad $\epsilon = 1 - P$ se selecciona una acción aleatoriamente.

- $Softmax$: en donde la probabilidad de selección de cada acción depende de su valor estimado. La más común sigue una distribución de Boltzmann o de Gibbs, y selecciona una acción con la siguiente probabilidad:

$$P = \frac{e^{Q(s_t,a_t)/T}}{\sum_{b=1}^{n} e^{Q(s_{t+1},a_{t+1})/T}} \tag{2.2}$$

  donde T es un parámetro positivo.

El esquema de selección de acciones que realiza un mejor balance entre explotación y exploración es ϵ-*greedy* (Morales y Sammut 2004).

Para conocer la función Q en el trabajo presentado por (Watkins 1989) se desarrolló un algoritmo recursivo llamado *Q-learning*. En pseudo-código de *Q-learning* viene descrito en el algoritmo 2.1.

Los parámetros para la actualización de la función Q son:

- α (factor de aprendizaje): sus valores se encuentran entre 0 y 1. Cuando vale 0 significa que los valores Q no se actualizan nunca y como consecuencia, no se aprende nada. Cuando toma valores cercanos a 1, el aprendizaje ocurre de forma rápida conservando los valores Q más actuales.

- γ (factor de descuento): sus valores se encuentran entre 0 y 1. Establece la importancia de las recompensas futuras en comparación con las inmediatas.

Para ilustrar el proceso de generación de políticas a través de *Q-learning*, se muestra un ejemplo de la aplicación de este algoritmo para el aprendizaje de tareas de navegación con un robot móvil.

Algoritmo 2.1 Algoritmo Q-learning.

Entrada S: conjunto de estados del ambiente.

A: conjunto de posibles acciones del agente.

r: conjunto de valores escalares de recompensa.

Salida Q: función que determina la política de toma de acciones del agente.

Inicializa $Q(s_t, a_t)$ arbitrariamente

Repite

Inicializa s_t

Para cada paso del episodio **hacer**

Selecciona una acción a_t en s_t siguiendo alguna política de selección de acciones (e.g., ϵ-$greedy$)

Ejecuta acción a_t, observa r_{t+1}, s_{t+1}

$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$

$s_t \leftarrow s_{t+1}$

Fin para

Hasta que s_t sea terminal

Ejemplo *Q-learning*

En este ejemplo, el modelo de aprendizaje por refuerzo consta de:

- Un conjunto finito de estados $S = \{(0,0), (0,1), (0,2), (1,0), (1,1), (1,2)\}$ que el agente percibe e identifica a través de sus sensores. En este caso el ambiente mide 3.0 $m.\times$ 2.0 $m.$ y fue discretizado en estados de 1.0 $m.\times$ 1.0 $m.$

- Un conjunto finito de acciones del agente $A = \{der$ (avanzar a la derecha), izq (avanzar a la izquierda), ar (avanzar hacia arriba), ab (avanzar hacia abajo)$\}$.

- Un conjunto finito de valores de recompensa $r = \{$100 una vez en el estado final, 0 en cualquier otro caso$\}$.

El objetivo es que el agente o robot (R) genere una política de navegación que le permita desplazarse desde cualquier estado al estado (0,2) en el mapa de celdas de la figura 2.1.

Figura 2.1: Mapa de celdas simple.

La política que se sigue para la elección de acciones en este caso es *greedy* y el valor del factor de aprendizaje (α) es 0,9 y el valor del factor de descuento (γ) es 1,0. La secuencia de acciones en la primera iteración del algoritmo de *Q-learning*, presentado anteriormente, se muestra en la figura 2.2.

En este caso, el robot se encuentrá en el estado (0,0), de forma aleatoria eligió, de sus cuatro acciones posibles, ejecutar la acción de desplazarse a su derecha llegando a (0,1) y en ese nuevo estado eligió de forma aleatoria, de sus cuatro acciones posibles, desplazarse nuevamente a su derecha llegando al estado final (0,2).

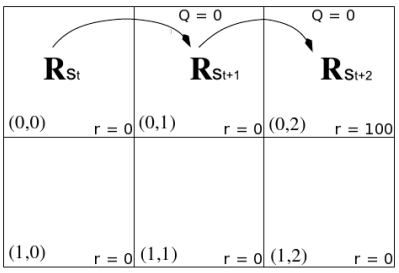


Figura 2.2: Ejemplo *Q-learning* primera iteración.

Los valores de la función Q fueron inicializados en 0 y sus valores, en la primera iteración de *Q-learning*, se actualizaron de la siguiente forma:

- El estado inicial es $s_t = (0, 0)$, la acción seleccionada es $a_t = der$, el estado subsecuente es $s_{t+1} = (0, 1)$, el valor de recompensa es $r_{t+1} = 0$, los valores de la función Q resultantes son:
  $Q((0, 0), a_{der}) = r((0, 0), a_{der}) + 0.9 \cdot \max\{Q((0, 1), a_{t+1} = der), Q((0, 1), a_{t+1} = izq), Q((0, 1), a_{t+1} = ab)\} = 0 + 0.9 \cdot \max\{0, 0, 0\} = 0.$

- El nuevo estado del ambiente es $s_{t+1} = (0, 1)$, la acción seleccionada es $a_{t+1} = der$, el estado subsecuente es $s_{t+2} = (0, 2)$, el valor de recompensa es $r_{t+2} = 100$, los valores de la función Q resultantes son:
  $Q((0, 1), a_{t+2} = der) = 100 + 0.9 \cdot \max\{0\} = 100.$

Una segunda iteración del algoritmo *Q-learning* se muestra en la figura 2.3. En este caso el robot comienza en el estado (1,1), elige de forma aleatoria desplazarse hacia arriba llegando al estado (0,1), posteriormente elige desplazarse a su derecha llegando al estado (0,2). Los valores de la función Q de la segunda iteración se actualizan de la siguiente forma:

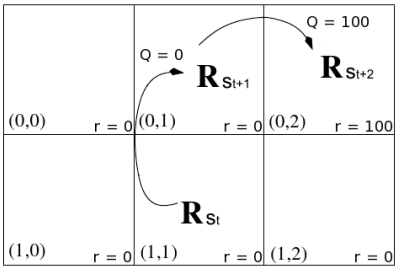


Figura 2.3: Ejemplo *Q-learning* segunda iteración.

- El estado inicial es $s_t = (1, 1)$, la acción seleccionada es $a_t = ar$, el estado subsecuente es $s_{t+1} = (0, 1)$, el valor de recompensa es $r_{t+1} = 0$, los valores de la función Q resultantes

son:

$Q((1, 1), a_{ar}) = r$ Q$((1, 1), a_{t+1} = ar) = 0 + 0.9 \cdot \max\{0, 0, 100\} = 90.$

- El nuevo estado del ambiente es $s_{t+1} = (0, 1)$, la acción seleccionada es $a_{t+1} = der$, el estado subsecuente es $s_{t+2} = (0, 2)$, el valor de recompensa es $r_{t+2} = 100$, los valores de la función Q resultantes son:
  Q$((0, 1), a_{t+2} = der) = 100 + 0.9 \cdot \max\{0\} = 100.$

Después de varias iteraciones, la función Q converge, i.e., todos los estados son visitados muchas veces y diferentes acciones aleatorias (del conjunto de posibles acciones) se ejecutan en cada uno de ellos. Los valores resultantes de la función Q son los que se muestran en la figura 2.4.

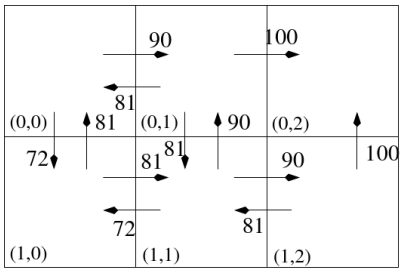


Figura 2.4: Ejemplo *Q-learning* al converger.

De esta forma, siempre que el robot se encuentre en determinado estado, debe seleccionar aquella acción que le otorgue el mayor valor Q para así llegar a su punto de destino o meta. Así, sin importar si se trata de un proceso determinista[1] o estocástico[2] para cada estado en que se llegue a encontrar el robot existirá la correspondiente acción que lo llevará al estado final o de recompensa.

[1]Ejecutar la misma acción en un estado siempre producirá el mismo estado subsecuente.

[2]Ejecutar la misma acción en un estado no siempre produce el mismo estado subsecuente debido a la incertidumbre en las acciones del robot.

Por ejemplo, en un proceso determinista, si el robot se encuentra en el estado (1,2) debe ejecutar la acción *ar* (arriba) para, con toda seguridad, llegar al estado final (0,2). En un proceso estocástico si el robot se encuentra en el estado (1,2) y ejecuta la acción *ar*, es altamente probable que llegue al estado final (0,2). Sin embargo, con cierta probabilidad, puede que el robot en lugar de llegar a (0,2) llegue al estado (0,1) o al (1,1) o inclusive, permanezca en (1,2). Sin embargo, para cualquiera que sea el estado al que llegue, la política siempre le indicará qué acción debe realizar para llegar al estado final.

La gran ventaja de *Q-learning* es la simplicidad del algoritmo, es sencillo de implementar, muestra buenos resultados para muchas aplicaciones en robótica, no requiere de un modelo previo del ambiente y toma en cuenta la incertidumbre en las acciones del robot.

Como ya se mencionó anteriormente, una de las principales desventajas de *Q-learning* es el tiempo que toma evaluar todas las posibles combinaciones de pares estado-acción en especial para espacios de estados muy grandes con miles o cientos de miles de posibles estados. Para hacer frente a esta desventaja se disminuye el espacio de búsqueda proporcionando sólo un subconjunto de estados y acciones relevantes para el aprendizaje de la tarea, es decir, partiendo del ejemplo de *Q-learning* presentado previamente, si en lugar de considerar cuatro acciones del robot por cada estado se pudieran considerar una o dos, el proceso de generación de de la política tomaría menor tiempo. Intentar disminuir el espacio de estados es una idea que se ha intentado hacer haciendo uso de diversas técnicas como son árboles de decisión (Cocora et al. 2006), macros (Torrey et al. 2008), procesos de *clustering* (Mahadevan y Connell 1991) y recientemente una de las técnicas que ha cobrado más fuerza debido a su simplicidad y buenos resultados es la clonación de comportamiento (Bratko et al. 1998, Morales y Sammut 2004, Govea y Morales 2007).

2.2 Clonación de comportamiento

La clonación de comportamiento (Bratko et al. 1998) es el método a través del cual las habilidades de los humanos, al ejecutar alguna tarea, son registradas y reproducidas por programas de computadora. La idea general es que si los humanos son capaces de realizar una tarea, en lugar de solicitarles que expliquen cómo la hacen, se les pida que la hagan para posteriormente producir una descripción simbólica de la tarea. Esta descripción simbólica, por lo general, son pares estado-acción como aquellos utilizados por aprendizaje por refuerzo para el desarrollo de políticas de control (Bratko et al. 1998, Morales y Sammut 2004).

Mientras la persona realiza la tarea que se quiere aprender, sus acciones son registradas junto con la situación que dio origen a tal acción. A este registro o ejemplo de tarea se le conoce como traza. La clonación de comportamiento, i.e., las trazas, son ejemplos de tareas que se quiere que el agente aprenda. Cada traza ($\tau_i \in T \mid T =$ conjunto de trazas o ejemplos de la tarea a aprender) es una secuencia de observaciones o de $frames$ registrados a lo largo del tiempo que dura la tarea:

$$\tau_i = \{f_{i_t}, f_{i_{t+1}}, f_{i_{t+2}}..., f_{i_{t+n}}\} \tag{2.3}$$

dónde f_{i_t} es el $frame$ de la i-ésima secuencia en el tiempo t.

Una observación o $frame$ corresponde a una lectura del conjunto de sensores con los cuales está dotado el equipo o dispositivo que utiliza la persona para generar las trazas, en robótica puede ser, una lectura de los sensores láser, sonares y odómetro del robot.

Para obtener las descripciones simbólicas a partir de las trazas se determina una interpretación a los valores de cada uno de los $frames$ que las trazas contienen. Es decir, para cada observación o $frame$ f_{i_t} se genera un par estado-acción.

Por ejemplo, en robótica, si cada observación corresponde a un registro con las lecturas de los sensores láser y sonares del robot así, como los valores del odómetro, es posible generar

una descripción simbólica para esta observación. Se toman los valores de los sensores láser y sonares para caracterizar el estado, i.e., determinar qué elementos rodean al robot. Y además se toman los valores del odómetro para determinar la acción que realizó el robot en dicho estado.

Estas representaciones simbólicas son dadas como entrada a algún programa de aprendizaje (e.g., aprendizaje por refuerzo). El programa de aprendizaje genera como salida un conjunto de reglas o políticas que reproducen el comportamiento o habilidades para ejecutar una tarea. Debido a que las trazas se generan por personas que conocen la tarea, sólo un subconjunto de estados y acciones relevantes para el aprendizaje de tarea son observados. De esta forma se logra disminuir el número de estados y de acciones por estado que los programas de aprendizaje deben tomar en cuenta para aprender a ejecutar la tarea. En seguida se muestra un ejemplo de clonación de comportamiento.

Ejemplo clonación de comportamiento

Para acelerar el proceso de generación de políticas de navegación, hecho con *Q-learning*, y que fue mostrado anteriormente (sección 2.1.1), se hará uso de clonación de comportamiento.

Para ello es necesario generar, ya sea en simulación o con un robot real, trazas con ejemplos de la tarea que se quiere aprender. Como en este caso se trata de una tarea de navegación se deben generar trazas o ejemplos que contengan diversas tareas de navegación como las mostradas en las figuras 2.5, 2.6 y 2.7.

Una vez generadas las trazas y almacenados los valores de cada par estado-acción correspondientes a cada frame, se aplica aprendizaje por refuerzo. Cada vez que el robot llegue a un estado s_t, antes de seleccionar una acción a_t de todo el conjunto de acciones A lee de sus trazas todos aquellos pares estado-acción que contengan a s_t. Por ejemplo, si $s_t = (0,1)$, se leen de las trazas todos aquellos pares estado-acción cuya descripción de estado sea igual a s_t, en este caso son, de la traza no. 1 el $frame$ 3 cuyos valores son $\{(0,1)\text{-}ab\}$ y de la traza no. 3 el $frame$ 3 cuyos valores son $\{(0,1)\text{-}ar\}$. Así en lugar de seleccionar una acción de

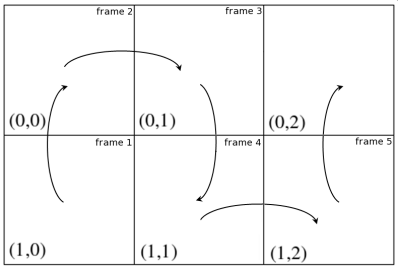


Figura 2.5: Ejemplo no. 1 de traza de navegación.

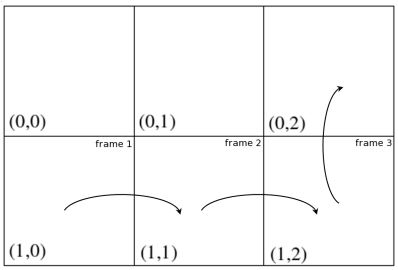


Figura 2.6: Ejemplo no. 2 de traza de navegación.

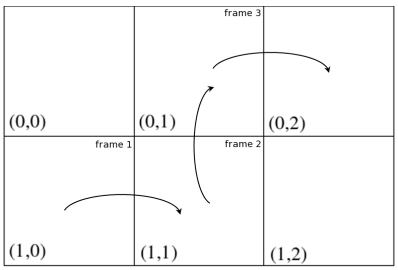


Figura 2.7: Ejemplo no. 3 de traza de navegación.

entre las cuatro acciones posibles de A, se selecciona una acción de entre un subconjunto de A con dos posibles acciones: ab y ar. Es decir, en lugar de seleccionar ejecutar una acción a_t ϵ A, se selecciona una acción a_t ϵ A' $|A' \subseteq A$.

Desafortunadamente, las acciones de las políticas de control de aprendizaje por refuerzo

son discretas, es decir, las acciones sólo pueden tomar un conjunto predefinido y finito de valores los cuales no pueden cambiar a través del tiempo. Una forma en la cual se puede permitir la ejecución de acciones continuas a *Q-learning* es realizando una regresión sobre los valores de las acciones discretas.

2.3 Regresión pesada local

En la figura 2.8, si el robot se encontrara en el estado (1,1) tiene dos posibles secuencias de acciones a seguir (indicada por flechas continuas) para llegar al estado final (0,2). Basta con que elija alguna de las dos secuencias de acciones para alcanzar tal estado. Sin embargo, lo mejor sería tener la capacidad de ejecutar una acción que lo lleve de manera más directa (indicada por una flecha punteada) al estado (0,2) pero, tal acción nunca fue definida en el conjunto A de todas las posibles acciones del robot. Así que algo de lo que se puede hacer para generar estas acciones continuas es aprovechar la información o valores de todas las acciones discretas que se pueden ejecutar en el estado (1,1). Esto se puede realizar a través de regresión pesada local.

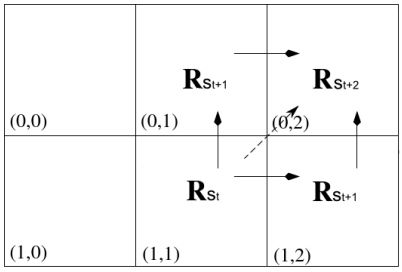


Figura 2.8: Acción ideal del robot para llegar al estado final.

La regresión pesada local es una generalización que construye una función ($\hat{f}(x_q)$) que ajusta los datos de entrenamiento ($y_{q1}, y_{q2}, ..., y_{q1}$) que están en la vecindad de un punto determinado x_q. Para ello se pueden usar diversos tipos de funciones como son lineales,

cuadráticas, cúbicas, etc. Si se utiliza una función lineal para hacer la generalización se utiliza la siguiente ecuación:

$$\hat{f}(x_q) = w_0 + w_1 * y_{q1} * x_q + \ldots + w_n * y_{qn} * x_q \tag{2.4}$$

Para realizar la regresión pesada local, es necesario determinar el peso o influencia que tendrá cada dato de entrenamiento $(y_{q1}, y_{q2}, ...y_{qn})$ sobre $\hat{f}(x_q)$, es decir, determinar los valores $w_0, w_1, ...w_n$.

2.3.1 Función de distancia

Para determinar el peso que se le va a asignar a cada dato de entrenamiento, es necesario calcular la distancia a la cual se encuentran dichos datos con respecto al punto sobre el cual se realiza la regresión (x_q). Mientras más alejado se encuentre el punto de entrenamiento y_{qi}, menor será su influencia sobre $\hat{f}(x_q)$. Para obtener ello se han desarrollado diversas funciones de distancia. La función típica para datos continuos es la Euclidiana

$$d_E(\mathbf{x_q}, \mathbf{x'_q}) = \sqrt{\sum_j (\mathbf{x_q}_j - \mathbf{x'_q}_j)^2} = \sqrt{(\mathbf{x_q} - \mathbf{x'_q})^T (\mathbf{x_q} - \mathbf{x'_q})} \tag{2.5}$$

dónde x_q es el punto sobre el cual se realiza la regresión y x'_q es el punto respecto al cual se calcula la distancia.

A pesar de que exisen más funciones de distancia, la función Euclidiana es de las más comunes, de las más sencillas, presenta muy buenos resultados y es la que será utilizada en el presente trabajo de tesis.

2.3.2 Función de pesos o *kernel*

Una vez calculada la distancia de todos los puntos de entrenamiento $(y_0, y_1, ...y_n)$ respecto al punto sobre el cual se realiza la regresión (x_q), se determina el peso que tendrá cada valor

y_i en $\hat{f}(x_q)$. Las funciones de peso deben de ser máximas a distancia cero y decaer suavemente con la distancia. La función de peso más común, debido a su sencillez y buenos resultados es el Kernel Gaussiano:

$$K(d) = exp(-d^2) \tag{2.6}$$

A través de esta regresión pesada local es posible generar funciones continuas a partir de una secuencia de puntos o valores discretos los cuales sirven de entrenamiento a la regresión. Estas funciones son las que determinan el conjunto de acciones continuas o trayectorias que debe seguir el robot para completar su tarea.

Ejemplo regresión pesada local

En la figura 2.8 el robot tiene dos posibles acciones (*ar* y *der*) a ser ejecutadas en el estado (1,1). Ambas acciones tienen el mismo valor Q así que la selección de cualquiera de las dos acciones es aleatoria. Supongamos que la acción que elige el robot es la de girar 0° y avanzar hacia la derecha (*der*), estos valores se asignan suponiendo que el robot se encuentra orientado hacia la derecha (figura 2.9 indicada por una flecha continua).

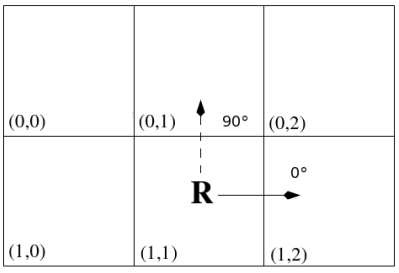


Figura 2.9: Acción discreta seleccionada por el robot siguiendo la política generada previamente por *Q-learning*.

El valor de la acción no seleccionada (figura 2.9 indicada por una flecha punteada), es decir, girar a la izquierda 90° y avanzar hacia arriba (*ar*) será utilizado para realizar la

regresión pesada local que permita generar las acciones continuas. Este valor de la acción no seleccionada será empleado como sigue; la distancia (Manhattan) d_M que existe entre el estado (1,1) y el estado (0,1) es de 1. Por lo tanto el peso w que se asigna a la acción del estado (0,1) es el siguiente:

- $K(d_M) = w = exp(-1^2) = 0.3679$

Entonces el valor de la regresión pesada local quedaría:

- $\hat{f}(x_q) = w_0 + w_1 * y_{q1} * x_q = 1 + 0.3679 * 90 * 1 = 34.1091$, donde w_0 = peso del estado (1,1), w_1 = peso del estado (0,1) y y_{q1} = valor de la acción no seleccionada.

Este valor de la regresión pesada local $\hat{f}(x_q)$ se suma al valor de la acción elegida que anteriormente era girar 0° y avanzar, entonces ahora la correspondiente acción continua resultante es girar 33.1091° y avanzar (figura 2.10).

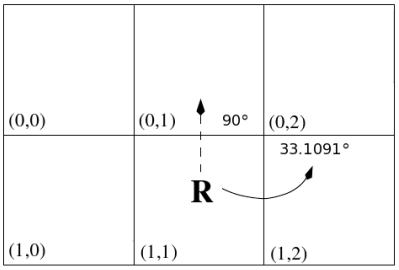


Figura 2.10: Acción continua generada por el robot a través de regresión pesada local.

En este capítulo se presentaron las bases necesarias para la comprensión de los términos empleados en la tesis. En particular se abordó aprendizaje por refuerzo a través de *Q-learning* por ser uno de los algoritmos más utilizados para el desarrollo de políticas de control en robótica. Se presentó el algoritmo de *Q-learning* y se mostró un ejemplo de generación de políticas de navegación a través de este algoritmo. Se mostró cómo es posible abordar dos de los cuatro principales problemas que tienen las técnicas de aprendizaje por refuerzo.

Clonación de comportamiento se mostró como una técnica que permite disminuir el espacio de búsqueda de *Q-learning* a través de ejemplos o trazas proporcionadas por el usuario. Regresión pesada local se mostró como una técnica de aproximación de funciones que permite que las acciones discretas de las políticas que genera aprendizaje por refuerzo puedan ser transformadas a acciones continuas.

A través de estas técnicas y algoritmos se desarrolló un método que permite generar políticas de control relacionales con acciones continuas a través de aprendizaje por refuerzo. Este método se describirá a detalle en el capítulo 4 del presente trabajo.

En el siguiente capítulo se presenta el Trabajo Relacionado. Este muestra los trabajos más relevantes, encontrados en la literatura, de desarrollos que permiten generar modelos o políticas de control para robots móviles. Esto se hace con la finalidad de distinguir las principales diferencias y aportaciones del método propuesto respecto a los métodos existentes.

3

Trabajo Relacionado

En el presente capítulo se exponen los principales métodos que se han encontrado para el aprendizaje de tareas en robótica. El estado del arte hace énfasis en aprendizaje por refuerzo ya que ese es el principal tema del presente trabajo. Sin embargo, se presentan los desarrollos encontrados más más relevantes en otras áreas de aprendizaje computacional orientado al control de robots.

3.1 Aprendizaje por refuerzo en robótica

En robótica móvil se han hecho importantes desarrollos de aprendizaje, control y automatización de tareas a través de aprendizaje por refuerzo. A continuación se presentan los trabajos más relevantes enfocados al aprendizaje de políticas de control para robots móviles, a través de aprendizaje por refuerzo.

En (Mahadevan y Connell 1991) un robot móvil aprende a transportar grandes cajas durante largos periodos de tiempo. Aprender a empujar objetos es un problema difícil y bien conocido en robótica, caracterizado por la incertidumbre en los resultados de las acciones del robot. En este trabajo se utilizó *Q-learning* en conjunto con una técnica de *Clustering*.

Esta técnica tiene por objetivo tomar como entrada grandes cantidades de información, proveniente de los sensores del robot y producir como salida un *cluster* que caracterice el estado del ambiente. De esta forma se disminuye la entrada de datos de los sensores del robot reduciendo la cantidad de información a procesar. El desempeño del robot compitió con el desempeño de una solución programada "a mano". Otro aspecto de este trabajo es que la tarea completa se dividió en sub-tareas y para cada sub-tarea se aprendió una política utilizando el enfoque propuesto. La desventaja de este método es el tiempo que toma realizar el proceso de *clustering* para cada estado en el que se encuentre el robot. Además, por cada estado distinto es necesario generar un nuevo *cluster* y, finalmente, se emplean acciones discretas para las maniobras del robot.

En (Schaal y Atkeson 1991) se construyó un robot con dos brazos que aprende a no dejar caer un artefacto, en forma de vara (llamado *devil-stick*), el cual tiene que ser pasado de un brazo a otro. Esta tarea involucra un espacio de estados en seis dimensiones y requiere que se determine, en un tiempo menor a $200msec.$, la decisión de las acciones del robot para evitar que el objeto caiga. Después de 40 intentos iniciales, el robot aprende a evitar que este artefacto caiga al piso por periodos largos de tiempo. Una variante del algoritmo de programación dinámica fue utilizada para mejorar gradualmente la política. Debido a la alta velocidad de respuesta que exigió el sistema, el espacio de búsqueda se acotó a un número muy pequeño de estados y de acciones las cuales, también eran discretas. Otra desventaja de este método es que se requiere conocer el modelo de transición de estados para ser proporcionado al algoritmo de programación dinámica.

En (Morales 2005) se utiliza aprendizaje por refuerzo para aprender a volar (en simulación) un avión de alto desempeño. El espacio de búsqueda es muy grande debido al tamaño del espacio de vuelo y a la cantidad de acciones que puede realizar el avión para mantener sus posiciones de elevación y de giro. Es por esto que el conjunto de estados y acciones que alimentan al proceso de aprendizaje por refuerzo son obtenidos a partir de ejemplos de trazas de vuelo. El método de aprendizaje por refuerzo presentado en este trabajo fue capaz de

generar políticas aplicables a tareas de vuelo que no habían sido vistas durante el proceso de aprendizaje, i.e., que no fueron vistas en los ejemplos de las trazas. Las políticas generadas también fueron capaces de volar al avión en circunstancias de vuelo distintas a las vistas durante el aprendizaje por variaciones de fuerza del viento y turbulencias. A pesar de que en este trabajo se logró desarrollar un método que permite librar situaciones no vistas durante el aprendizaje y se reduce el espacio de búsqueda a través de clonación de comportamiento, al igual que con los métodos anteriores, las acciones son discretas.

En (Lee. et al. 2006) se aplicó aprendizaje por refuerzo para el aprendizaje de tareas de un robot cuadrúpedo, principalmente, la evasión de obstáculos. Para ello se realizó una descomposición de la tarea en dos diferentes niveles jerárquicos. En el primer nivel se selecciona una configuración específica de las extremidades del robot. En el segundo nivel se generan movimientos continuos para mover cada extremidad del robot a una posición específica. El control del primer nivel utiliza un *MDP* para lograr su objetivo. Un esquema similar, con *MDP*'s jerárquicos, se presentó en (Cuaya y Munoz 2007). Aquí se desarrolló un modelo para el control de un robot hexápodo hexagonal. En este trabajo se utiliza un *MDP* formado a su vez por 2 *MDP*'s que controlan acciones básicas del robot, como levantar o apoyar una pata, y un tercer *MDP* que coordina a los anteriores. De esta manera lograron atacar problemas de control de navegación, sincronización y movimiento. Debido al esquema de *MDP*'s jerárquicos que utilizan ambos trabajos, los robots son capaces de enfrentar y librar situaciones no vistas durante el proceso de aprendizaje. Sin embargo, en ambos trabajos se requieren definir los modelos de transición de los *MDP*'s y se debe definir su estructura jerárquica.

Otros desarrollos interesantes de aprendizaje por refuerzo se han dado en robótica colectiva o cooperativa. En (Mataric 1994) se describe un experimento con cuatro robots moviles los cuales deben llevar a cabo tareas de recolección de objetos (pequeños discos) en un espacio cerrado. Una vez recolectado el objeto, éste debe ser descargado en un sitio predefinido. Este esquema es importante debido a tres aspectos fundamentales: (i) la tarea fue dividida en subtareas, (ii) cada robot aprendió su propia política sin alguna comunicación explícita

con los demás robots, y (iii) el espacio de estados fue reducido drásticamente a un pequeño número de estados discretos los cuales eran caracterizados por un pequeño conjunto de variables *booleanas*. El desempeño de las políticas fue casi tan bueno como la solución programada de forma explícita al medir la cantidad de discos recolectados y transportados correctamente a su destino.

El trabajo presentado en (Stone 1997) utiliza aprendizaje por refuerzo para que los miembros de un equipo de fut-bol de robots (presentados en *RoboCup Soccer*), fueran capaces de aprender a qué miembro del equipo pasar el balón. Cada robot debía generar su propia política tomando en cuenta las observaciones (configuraciones o estados) de los compañeros y atacantes. Cuando el balón se pasaba de forma correcta a un miembro del equipo se recibía recompensa positiva, cuando el balón no era pasado de forma correcta (no llegaba a su compañero) o era pasado a un rival, se recibía refuerzo negativo. De esta forma los robots aprendieron a hacer los pases de balón a compañeros sin atacantes o rivales cerca y aprendieron a realizar los pases de forma correcta.

En general, las técnicas de aprendizaje por refuerzo presentan las siguientes ventajas:

- Han sido utilizados para el aprendizaje de políticas de control para diversas tareas con robots móviles reales.
- No se requiere programación explícita del robot ya que sólo se requiere de una función o valor de recompensa que indique al robot qué acciones son mejores que otras.
- Los algoritmos de aprendizaje por refuerzo son sencillos de implementar.

entre sus desventajas se pueden encontrar:

- Las acciones de los robots son discretas.
- Las políticas que genera el esquema clásico de aprendizaje por refuerzo no son transferibles.

- El tiempo que toma generar las políticas de control depende del tamaño del espacio de estados, tareas que tienen decenas o cientos de miles de estados con varias acciones posibles por estado requieren de mucho tiempo, inclusive días para evaluar todas las combinaciones posibles.

Contribución. En el presente trabajo se presenta un algoritmo de aprendizaje por refuerzo (*rQ-learning*) que permite que los estados y las acciones sean representados de forma relacional, haciendo que las políticas generadas con este algoritmo sean relacionales, es decir, sean transferibles o re-utilizables entre dominios similares (tipo oficina u hogares). Para disminuir el tiempo de generación de las políticas de control se utiliza clonación de comportamiento lo que permite disminuir el espacio de estados a través de ejemplos de la tarea que se quiere aprender. Para hacer que las acciones que ejecuta el robot sean continuas, se utiliza un esquema de regresión pesada local que utiliza la información de clonación de comportamientopara transformar las acciones discretas de la política en acciones continuas.

3.2 Representaciones Relacionales

Recientemente se ha observado un creciente interés en la trasferencia o re-utilización de conocimiento de tareas previamente aprendidas. Esto se debe a que la capacidad para inferir de forma autónoma el conocimiento necesario para enfrentar tareas no vistas en las etapas de aprendizaje, proporciona robustez y adaptabilidad a los sistemas. Como se ha visto, en varios de los trabajos presentados anteriormente se han utilizado diferentes esquemas para permitir la transferencia de conocimiento. Otra de las principales estrategias propuestas es incorporar representaciones relacionales a los enfoques de aprendizaje.

En (Dzeroski et al. 1998) se presenta el método de aprendizaje por refuerzo relacional (*Relational Reinforcement Learning*, *RRL*). Éste es un método que combina aprendizaje por refuerzo con programación lógica inductiva (*Inductive Logic Programming*, *ILP*). Esto se hace

con la finalidad de desarrollar políticas de control relacionales mientras el agente interactúa con su ambiente. El aprendizaje por refuerzo relacional genera políticas expresadas con una representación relacional utilizando predicados predefinidos. Estos predicados predefinidos describen el conjunto de estados del ambiente. Para cada uno de estos estados se tiene un conjunto predefinido de posibles acciones (discretas). Posteriormente, aprendizaje por refuerzo es aplicado a este conjunto de estados y acciones para generar la política de control.

En (Smart 2002), los autores construyen modelos del ambiente en forma de *MDP*'s. Utilizan programación lógica inductiva para aprender definiciones lógicas de la región del espacio de búsqueda que llevan a estados con altas recompensas cuando se ejecuta alguna acción. Estas definiciones lógicas aprendidas son concatenadas para formar un *MDP* que, de forma abstracta, representa a todo el dominio.

En (Klingspor et al. 1996) se desarrolló un método para que el robot pueda aprender conceptos relacionales como *in_front_of_door*, *along_door*, *move_closer_to_wall*, *line*, *concave*, *convex* en diferentes niveles de abstracción (5 niveles en total). Posteriormente estos conceptos fueron utilizados para controlar al robot. Sin embargo, debido a la complejidad (cantidad de combinaciones posibles y estructura jerárquica) de los diferentes niveles de abstracción el robot no pudo llevar a cabo, satisfactoriamente, las tareas de navegación que le fueron solicitadas.

En (Torrey et al. 2008) se presenta un método para construir macros relacionales para la transferencia de conocimiento. Estas macros utilizan lógica de primer orden para la toma de decisiones. Una macro consiste de un conjunto de nodos los cuales contienen reglas que determinan las transiciones entre nodos y las condiciones para la toma de decisiones. En este enfoque se utiliza información de ejemplos de aprendizaje como entrada de una versión modificada de *Aleph*. Con estos ejemplos, *Aleph* es capaz de inducir cláusulas lógicas las cuales son utilizadas para la representación de los estados y la generación de las reglas que determinan las transiciones así como para las reglas para la selección de acciones.

En (Cocora et al. 2006) se presenta una propuesta para generar árboles de decisión

relacionales, como estrategias abstractas de navegación a partir de trazas o ejemplos de entrenamiento. Un árbol relacional de decisión es un árbol binario. Cada nodo captura una prueba lógica. Si ésta tiene éxito, el subárbol izquierdo es analizado en las pruebas lógicas subsecuentes, de otra forma, se considera el subárbol derecho. Las hojas del árbol representan la acción a ser ejecutada. El método utiliza ejemplos de entrenamiento como entrada de una variante del método C4.5 de Quinlan. Este método utiliza los ejemplos para generar un árbol binario relacional. Estos ejemplos contienen conjuntos de estados del ambiente y las acciones ejecutadas en cada estado. La variante del método C4.5 es capaz de obtener las relaciones entre estos estados y acciones, y con ellos construir el árbol binario relacional.

Los enfoques anteriores utilizan representaciones relacionales para aprender una política de control y transferir el conocimiento aprendido o generado. Desafortunadamente, dichos métodos consideran solamente la ejecución de acciones discretas. Además, estos enfoques requieren de mucho tiempo de entrenamiento aún cuando algunos de ellos utilizan ejemplos de las tareas que se quieren aprender, como es el caso de los trabajos desarrollados en (Torrey et al. 2008) y (Cocora et al. 2006).

Contribución. En este trabajo se desarrolló una representación relacional que utiliza predicados de primer orden para describir los estados y las acciones del robot. Esta representación está basada en puertas, paredes, esquinas, habitaciones y obstáculos lo cual la hace más comprensible para los usuarios que los enfoques de árboles de decisión o de macros que normalmente se utilizan en este tipo de representaciones relacionales. Los estados y las acciones descritos de forma relacional con la representación que se propone en este trabajo de tesis serán dados como entrada al algoritmo de aprendizaje por refuerzo *rQ-learning* mencionado anteriormente, para que éste genere políticas de control relacionales con acciones discretas. El siguiente paso es transformar estas acciones discretas a continuas.

3.3 Clonación de Comportamiento

La clonación de comportamiento, i.e., ejemplos o trazas de tareas, resulta muy útil cuando se tienen que llevar a cabo procesos de aprendizaje de tareas en robots reales. Esto debido a que permiten reducir los grandes tamaños de los espacios de búsqueda generados por la cantidad de datos de los sensores de los robots o agentes y por los tamaños de los ambientes reales.

En (Morales 2005) se utilizan trazas de pilotos humanos para aprender a volar un avión de alto desempeño. Las trazas se utilizaron para reducir el enorme tamaño del espacio de búsqueda ocasionado por la cantidad de elementos de control con los que cuenta el avión y por el tamaño de los ambientes de vuelo. A partir de las trazas, el sistema fue capaz de generar pares de estados-acciones que describen las maniobras realizadas por los pilotos en diversas situaciones. En el trabajo se muestra la reducción del espacio de estados que se obtuvo al utilizar las trazas. Las políticas generadas fueron capaces de llevar a cabo misiones de vuelo que no fueron vistas en las trazas debido a la representación relacional de estados que se utilizó. Sin embargo, todo el conjunto de acciones que se utilizó en dicho trabajo fue discreto.

En (Govea y Morales 2007) se desarrolló un sistema llamado TOPSY (*Teleo-OPerator learning SYstem*) capaz de aprender conjuntos de reglas de control conocidas como teleo-operadores, i.e., *TOP's*, a partir de trazas de tareas en robótica. La información de las trazas es proporcionada a un sistema de programación lógica inductiva capaz de generar conceptos de alto nivel y *TOP's*. Con los *TOP's* generados a partir de las trazas, el robot es capaz de llevar a cabo las tareas que se realizaron con tales trazas (evadir obstáculos, orientarse rumbo a la posicón de destino o meta y dirigirse hacia ella). Debido a la representación relacional utilizada en ese trabajo, el robot es capaz de realizar sus tareas aún en presencia de situaciones no vistas en los ejemplos de entrenamiento. En este enfoque no se genera una

política sino una estructura gerárquica de reglas de control o *TOP's* que permiten determinar el conjunto de acciones continuas que debe ejecutar el robot.

En (Wang et al. 2003) se integran comandos de usuario, a diferentes niveles de abstracción, para permitir que un agente (un robot real) genere rápidamente una política de control. El enfoque utilizado presenta un método que logra cierta autonomía en el aprendizaje de las tareas combinando comandos de usuario con políticas de control generadas de forma autónoma en un modelo de un Proceso de Desición Semi-Markoviano (*SMDP*). Los comandos de usuario a altos niveles de abstracción son proporcionados al modelo del Proceso de Desición Semi-Markoviano en forma de sub-metas temporales a ser logradas, o como sugerencias de acciones específicas a ser ejecutadas. Estas entradas se utilizan para, temporalmente, dirigir al robot. Al mismo tiempo, los comandos de usuario sirven de entrada a un componente de aprendizaje, el cual optimiza la política de control autónoma para la tarea en proceso. Aquí *Q-learning* es utilizado para estimar la función de utilidad. La principal desventaja del método es el alto nivel de interacción que se requiere entre el usuario y el robot para aprender las tareas.

Un robot es tele-operado en (Conn y Peters 2007) para aprender secuencias de comportamiento las cuales lo llevarán a su objetivo. A través de estas secuencias, el robot genera pares estado-acción que describen la tarea que se está aprendiendo. Desafortunadamente, este método, al igual que el método anterior, consideran acciones discretas para llevar a cabo las tareas del robot.

Las trazas o guías proporcionadas por el usuario permiten el uso y aplicación de métodos de aprendizaje en robots móviles o de servicio reales. Esto se debe a que estas guías reducen los datos de entrenamiento del espacio de estados. Sin embargo, muchos de los enfoques que hacen uso de estas trazas sólo consideran la ejecución de acciones discretas.

En resumen, la clonación de comportamiento presenta las siguientes ventajas:

- Se aprenden reglas de control o tareas a partir de ejemplos que "cualquier" persona

puede proporcionar.

- Permite obtener reglas de comportamiento que difícilmente pueden programarse de forma manual.

- Los experimentos realizados con robots o agentes reales han mostrado buenos resultados aunque han sido limitados debido a las representaciones proposicionales utilizadas.

Es un método relativamente fácil de aplicar.
Por otro lado, presenta las siguientes desventajas:

- Por lo general los clones ejecutan acciones discretas.

- Los clones obtenidos pueden ser frágiles a pequeños cambios en los parámetros del ambiente.

- Es necesario obtener varios y diversos ejemplos de la tarea que se quiere aprender para obtener buenos resultados.

Contribución. En esta tesis se implementa un algoritmo de clonación de comportamiento en el que, a partir de ejemplos de tareas de navegación o de seguimiento con robots móviles, se aprenden pares estado-acción que permiten reproducir el comportamiento observado en tales ejemplos. Se utiliza la representación relacional en lógica de primer orden, mencionada anteriormente, para estos pares estado-acción lo cual hace a los clones más robustos que en el enfoque propocisional utilizado normalmente (Morales y Sammut 2004). Los objetivos principales de clonación son: (i) acelerar el proceso de generación de las políticas de aprendizaje por refuerzo, ya que los pares estado-acción que se obtengan de los ejemplos de las trazas reducen el espacio de búsqueda de la política de control, y (ii) proporcionar información de las acciones que ejecutó el usuario, durante los ejemplos de las trazas, para transformar las acciones discretas de aprendizaje por refuerzo en acciones continuas.

3.4 Acciones Continuas

Las acciones continuas, en comparación con acciones discretas, generan trayectorias que recorren menores distancias, son ejecutadas en menos tiempo, acumulan menos errores y son más seguras para la integridad del robot mismo y de los diversos elementos del ambiente. Por lo general, para permitir el uso de acciones continuas, lo que se realiza son diversas aproximaciones o estimaciones de los valores de la función Q de *Q-learning*.

En (Nguyen-Tuong y Peters 2008) se utilizan regresiones Gausianas (*Gaussian Regressions*, *GR*) para llevar a cabo tareas de manipulación de objetos con un brazo mecánico. La ejecución de las maniobras del brazo robot deben ser precisas, por tal razon, las acciones se afinan a través de estas regresiones. Lo que se tiene en ese trabajo es un conjunto de acciones predefinidas que el brazo robot puede realizar, a los valores de estas acciones predefinidas se les aplica la regresión Gaussiana. Al realizar la regresión se genera una función continua. A través de esta función continua el brazo del robot es capaz de mapear sus estados a acciones con lo cual se realizaron maniobras más suaves. Este método no fue aplicado a robots móviles.

En (Hasselt y Wiering 2007) se describe una nueva clase de algoritmos llamados *Continuous Actor Critic Learning Automaton* (*CACLA*). Este es un sistema de entrenamiento que extiende al método *Actor Critic Learning* con la finalidad de manejar problemas que involucran estados y acciones continuos.

El trabajo desarrollado en (Gaskett et al. 1999) es una red neuronal. El sistema descrito en este trabajo consiste de una red neuronal combinada con un nueva técnica de interpolación la cual aproxima los valores Q de las acciones. El trabajo presentado es capaz de generar una función continua en todo el espacio de búsqueda. Sin embargo, este enfoque no garantiza encontrar una solución debido a que se pueden presentar ciertos problemas de divergencia durante el proceso de generación.

Resumiendo, las principales ventajas de los métodos de aproximación de funciones continuas son las siguientes:

- Permiten la ejecución de acciones continuas lo cual hace que las maniobras o movimientos de los agentes sean:

  Más suaves.

  Más precisos.

  Más rápidos, al ser comparados contra acciones discretas.

Sus principales desventajas son:

- Estos métodos llevan implícito un alto costo computacional (Nguyen-Tuong y Peters 2008).
- Se necesitan largos tiempos de entrenamiento ya que tales métodos deben generar una función continua que cubra todo el espacio de búsqueda.
- Se pueden presentar problemas de divergencia en la búsqueda de las soluciones.
- Estas soluciones, por lo general, no son transferibles.

Contribución. Se propone un esquema de regresión pesada local que combina la información de las acciones del usuario de clonación de comportamiento, con la información de las acciones discretas de la política de control generada por *rQ-learning*. Esta combinación transforma las acciones discretas de la poítica de *rQ-learning* en acciones continuas y debido a la representación relacional utilizada, las soluciones, además de permitir la ejecución de acciones continuas, son transferibles. Este esquema de regresión pesada local no presenta problemas de divergencia ya que la regresión no se hace en todo el espacio de estados y tampoco consume grandes cantidades de tiempo ya que la regresión se realiza sobre un conjunto muy reducido de elementos, como se verá a detalle en capítulos posteriores.

En este capítulo se presentó el Trabajo Relacionado concerniente a aprendizaje por refuerzo y a los esquemas o métodos que se han desarrollado para permitir esquemas de control re-utilizables, que permitan la ejecución de acciones continuas y que busquen disminuir el espacio de estados con la finalidad de disminuir los tiempos de generación de las soluciones. De este análisis se puede extraer que, de manera general, los métodos clásicos de aprendizaje por refuerzo asumen un conjunto pequeño y discreto, tanto de estados como de acciones, para la generación de las políticas de control. Lo que comúnmente se hace para reducir la cantidad de estados y acciones es utilizar representaciones que permitan abstraer los estados y las acciones. De esta forma se puede escalar el método de aprendizaje por refuerzo a espacios de estados más grandes. Las representaciones relacionales permiten transferir políticas de control entre tareas o ambientes pero la mayoría de los métodos consideran acciones discretas. Los métodos para aproximar funciones permiten la ejecución de acciones continuas pero tienden a ser lentos o computacionalemnte caros y por lo general las políticas no son transferibles.

En el presente trabajo se transforma la información de bajo nivel de los sensores del robot en una representación relacional basada en puertas, habitaciones, paredes, esquinas y obstáculos la cual simplifica el espacio de estados y permite, a través de un algoritmo de aprendizaje por refuerzo llamado *rQ-learning*, generar políticas transferibles a dominios diferentes pero similares tipo hogar u oficina. También se utilizan trazas proporcionadas por el usuario para acelerar y simplificar el proceso de aprendizaje, con la finalidad de aplicar el método para tareas de aprendizaje en robots reales. Finalmente, se utiliza un esquema de regresión pesada local que utiliza la información de las trazas del usuario para convertir la política original con acciones discretas en una política con acciones continuas. El siguiente capítulo expone a detalle cada parte del método propuesto.

4

Método Propuesto

El método propuesto para generar políticas de control relacionales, que permiten la ejecución de acciones continuas a través de aprendizaje por refuerzo, se divide en dos fases. La primera fase emplea clonación de comportamiento, representaciones relacionales de estados y acciones, y aprendizaje por refuerzo para generar una política de control relacional con acciones discretas. Esta política es una primera aproximación con acciones burdas, hacia una política relacional con acciones continuas. Utilizando clonación de comportamiento la política se genera en pocas iteraciones y utilizando representaciones relacionales para caracterizar estados y acciones, las políticas generadas con aprendizaje por refuerzo pueden ser re-utilizadas en dominios o ambientes diferentes pero similares (tipo hogar u oficina con habitaciones, con puertas o accesos a las habitaciones, con paredes planas, con pasillos). La segunda fase refina las acciones burdas de esta política discreta a través de regresiones pesadas locales, generando así una política relacional con acciones continuas.

Las dos fases de este método, llamado Aprendizaje por Refuerzo Relacional con Acciones Continuas (*Two-Step Relational Reinforcement Learning with Continuous Actions, TS-RRLCA*), son detalladas en el presente capítulo.

4.1 Marcas naturales del ambiente

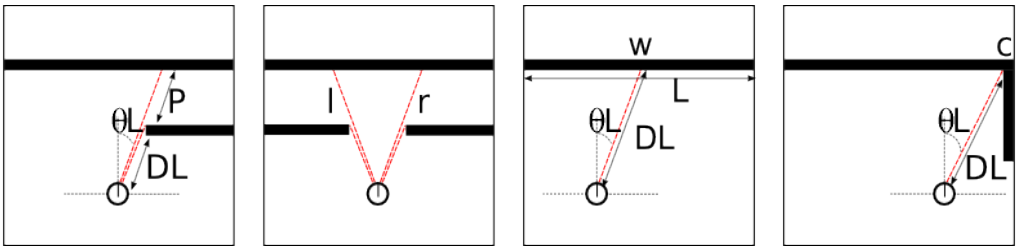
Cuando un robot móvil de servicio desempeña una tarea, registra y proporciona lecturas provenientes de sus sensores. Estos sensores pueden generar enormes cantidades de datos o mediciones, las cuales pueden saturar a los procesos de aprendizaje debido a la gran cantidad de información que deben procesar estas últimas. Con el objetivo de reducir la cantidad de información proveniente de los sensores láser y sonares del robot, se desarrolló un método para transformar las lecturas de dichos sensores (información de bajo nivel), en conjuntos más pequeños de descripciones más significativas. Este método se basa en el trabajo presentado por (Hernández y Morales 2006) y en el trabajo de (Herrera-Vega 2009).

En el trabajo de (Hernández y Morales 2006) se presentó un método para transformar las lecturas de los sensores láser del robot en información consistente de marcas naturales del ambiente. En el trabajo de (Herrera-Vega 2009) las lecturas de los sensores láser del robot se utilizan para identificar la ubicación local o estancia actual del robot.

Las marcas naturales del ambiente que se identifican son: (i) discontinuidades (figura 4.1(a)), definidas como una variación abrupta en las distancias registradas por dos sensores láser contiguos, (ii) esquinas (figura 4.1(d)), definidas como la posición en la cual dos paredes se unen y forman un ángulo y (iii) paredes (figura 4.1(c)), definidas como líneas en las lecturas del sensor láser las cuales son identificadas a través de una transformada de Hough que se realiza de forma local y de manera rápida. En el presente trabajo también se agregaron obstáculos (figura 4.1(e)), los cuales se identifican a través de los sonares del robot y se definen como cualquier elemento detectado dentro de cierto rango.

Una marca natural del ambiente se representa con una tupla compuesta por cuatro atributos: (DL , θL, A, T). DL y θL son, respectivamente, la distancia y la orientación relativas de la marca natural, con respecto al robot. T es el tipo de marca natural del ambiente: l para discontinuidad izquierda, r para discontinuidad derecha (figura 4.1(b)), c para esquina,

w para pared y o para obstáculo. A es un atributo distintivo cuyo valor depende del tipo de marca natural, para discontinuidades A representa la profundidad de la discontinuidad y para paredes A es la longitud de la pared. Para las otras marcas naturales, no se utiliza dicho atributo.

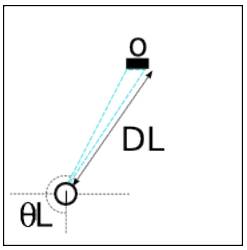


(a) Atributos discontinuidades.

(b) Tipos de discontinuidades.

(c) Atributos paredes.

(d) Atributos esquinas.

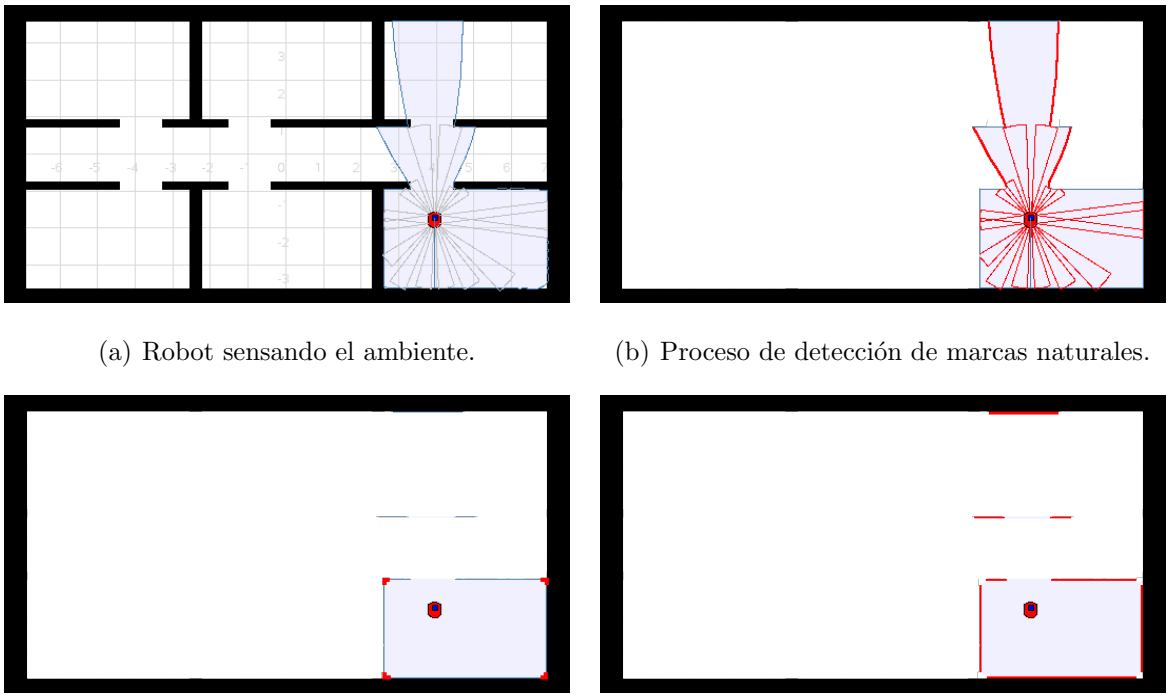


(e) Atributos obstáculos.

Figura 4.1: Ejemplos marcas naturales del ambiente y atributos asociados a ellas.

El proceso de extracción de marcas natuales se divide en tres pasos. El primer paso identifica discontinuidades en las lecturas de los sensores láser y obstáculos en las lecturas de los sonares, esto lo hace analizando las lecturas de estos sensores en contrasentido de las manecillas del reloj comenzando con el sensor que se encuentra en la parte trasera del robot. Basándose en las discontinuidades las lecturas de los láser se dividen en segmentos. El segundo paso identifica esquinas en estos segmentos. Basándose en estas esquinas, se dividen los segmentos en subsegmentos. El tercer paso toma los subsegmentos restantes y realiza una transformada de Hough de manera local para calcular los parámetros, i.e., obtener líneas,

asociadas a posibles paredes. El proceso descrito anteriormente se muestra en la figura 4.2.



(a) Robot sensando el ambiente.

(b) Proceso de detección de marcas naturales.

(c) Proceso de detección de marcas naturales.

(d) Proceso de detección de marcas naturales.

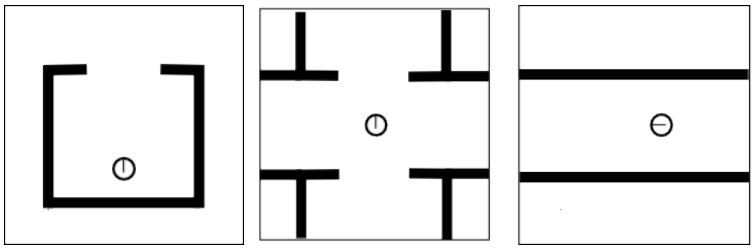
Figura 4.2: Proceso de detección de marcas naturales del ambiente.

Como posibles ubicaciones locales o estancias en las cuales se puede encontrar el robot se identifican: habitación, pasillo e intersección (sitio en dónde las habitaciones y los pasillos se intersectan). La ubicación local o estancia actual del robot se identifica proporcionando los valores de las lecturas de los sensores láser como entrada a un proceso de *clustering*. Este proceso toma los valores de las lecturas actuales del robot para clasificarlas en diversos *clusters*. Cada *cluster* corresponde a una estancia del ambiente del robot.

Para el entrenamiento o generación de los *clusters* se requiere un mapa métrico de $n * m$ celdas. En cada celda el robot sensa el ambiente con sus sensores láser. Por cada celda se obtiene un vector de descriptores geométricos. Los valores de los vectores son agrupados en n *clusters* utilizando *k-means*, donde n representa el número de ubicaciones locales o

estancias del robot. Una vez generados los *n clusters* se registran los valores de sus centros.

Para la clasificación, i.e., para identificar la ubicación local actual, cuando el robot está en el ambiente se obtienen las lecturas de sus sensores láser, se genera el vector de descriptores geométricos de estas lecturas y se calcula la distancia Euclidiana entre este vector y los valores de los centros de los *n clusters* para saber a cual pertenece. Las figura 4.3 muestra ejemplos de ubicaciones locales o estancias y la correspondiente clasificación proporcionada por este proceso.



(a) Habitación. (b) Intersección. (c) Pasillo.

Figura 4.3: Ejemplos de ubicaciones locales o estancias del robot.

La figura 4.5 muestra un ejemplo de la información que resulta luego de aplicar los métodos de detección de marcas naturales y de la ubicación del robot, presentados anteriormente, a las lecturas proporcionadas por los sensores láser y sonares del robot de la figura 4.4[1].

En el ejemplo anterior, en lugar de tener 180 valores de las lecturas del láser y 4 valores de las lecturas de los sonares, se tiene un registro con 16 valores el cual proporciona una mayor cantidad de información sobre los elementos que rodean al robot. Esta información conjunta de las marcas naturales del ambiente y la ubicación local o estancia actual del robot se utiliza para caracterizar los estados del ambiente del robot. Dicha caracterización se realiza a través de predicados de primer orden y es descrita a detalle en la siguiente sección.

[1]En este ejemplo se tomaron las lecturas del sensor láser correspondientes a los 180° de la parte frontal del robot y los cuatro sonares traseros colocados a -170°, -150°, 170° y 150°. Esta configuración de sensores en la que corresponde al robot real.

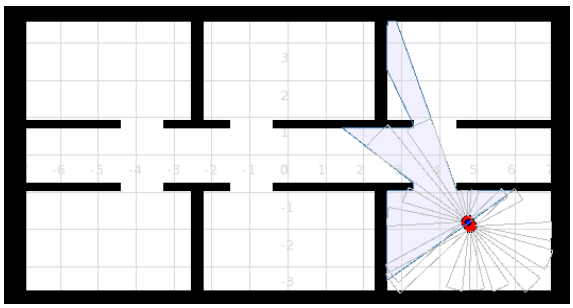
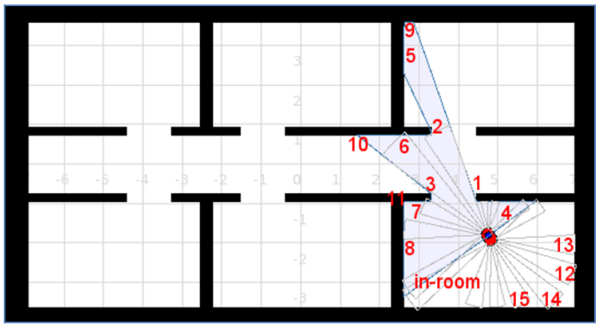


Figura 4.4: Robot sensando el ambiente a través de láser y sonares.



Ubicación
in-room

N	T	θL	DL	A
1	r	-17.6	0.92	4.8
2	l	-7.54	1.62	3.0
3	l	17.8	1.78	2.39
4	w	-35.7	0.87	1.51
5	w	-8.55	4.62	1.06
6	w	-6.54	2.91	1.88
7	w	23.63	1.73	0.53
8	w	53.8	2.13	2.38
9	c	-14.58	5.79	0.00
10	c	31.68	2.30	0.00
11	c	22.33	1.68	0.00
12	o	-170	1.87	0.00
13	o	-150	1.63	0.00
14	o	170	1.22	0.00
15	o	150	1.43	0.00

Figura 4.5: Ejemplo de marcas naturales del ambiente y de la ubicación local o estancia actual del robot.

4.2 Representaciones relacionales para caracterización de estados y acciones

Intentar aprender de forma directa basándose en los estados de todo el espacio de búsqueda, derivado de la información producida por el ambiente y por los sensores del robot es difícil

para cualquier enfoque de aprendizaje por refuerzo. Aún con una discretización burda del espacio de estados es muy fácil producir miles o millones de pares estado-acción[2]. Además, una vez que se genera una política para realizar una tarea particular, se debe aprender una nueva política si el objetivo o meta de la tarea cambian. Lo que se quiere es aprender políticas de control que puedan ser utilizadas para tareas con diferentes metas o puntos de destino y que puedan ser aplicables a diversos ambientes.

En el presente trabajo se desarrolló una representación relacional en la cual es fácil codificar la posición relativa del robot con respecto a su meta o con respecto a los elementos que se encuentran en el ambiente. La idea es representar a los estados como conjuntos de propiedades o relaciones que puedan ser utilizadas para caracterizar un estado particular y que pueden ser comunes a otros estados. Por ejemplo, cuando una persona pide direcciones para llegar a alguna habitación dentro de una nueva casa u oficina, las instrucciones que recibe no son: avanza $5.34m.$, posteriormente gira -62.5° y avanza $1.65m$. Sino que recibe información que le permite establecer relaciones entre los elementos u objetos del mundo que se encuentran a su alrededor, las cuales, generalmente, son más fáciles de identificar y de comprender. En este caso podría ser, avanza por el pasillo y en la primera puerta a la derecha. Este tipo de representaciones relacionales, además de ser más naturales o comprensibles para las personas que sólo números o lecturas crudas provenientes de un sensor, permiten transferir conocimiento entre dominios; ya que no importa si el pasillo mide $0.5m.$ o $500m.$ la persona no girará a su derecha hasta no ver la primera puerta.

Un estado relacional (r-estado), es una conjunción de predicados de primer orden. En robótica de servicio, estas relaciones pueden representar la distancia y la orientación del robot relativa a la meta o la distancia y orientación del robot respecto a puertas, a paredes, a obstáculos, etc. Por ejemplo el r-estado:

[2]Considere por ejemplo un espacio de 100 m^2. Una discretización burda de 0.25 m. × 0.25 m. con 4 acciones para el robot, dan un total $6,400$ posibles combinaciones de pares estado-acción

location(*in-room*),
goal_position([*right, close*]).

cubre todos aquellos estados en los cuales el robot se encuentra en una habitación y su meta se encuentra cerca y a la derecha.

En el presente trabajo de tesis, los estados son caracterizados por los siguientes predicados de primer orden. Estos predicados reciben como parámetro un registro (como el mostrado en la figura 4.5) que contiene marcas naturales del ambiente y la ubicación local o estancia actual del robot.

- *location*: Ubicación local o estancia del robot.
- *doors_detected*: Orientación y distancia del robot a puertas.
- *walls_detected*: Longitud, orientación y distancia del robot a paredes.
- *corners_detected*: Orientación y distancia del robot a esquinas.
- *obstacles_detected*: Orientación y distancia del robot a obstáculos.
- *goal_position*: Orientación y distancia del robot a la posición de destino o meta.
- *in_dest*: Indica si el robot ha llegado o no a su destino.

La forma en la cual se realiza el proceso de descripción de los estados (i.e., obtener las relaciones entre el robot y los elementos del ambiente) se describe a continuación.

4.2.1 Predicado *location*

Verifica que el valor de ubicación local o estancia actual del robot, recibido como parámetro, sea válido. Sus posibles valores son: *in-room*, *in-passage* e *in-intersection*.

4.2.2 Predicado *doors_detected*

Utiliza los valores de atributos tipo r y l (θL_r, DL_r, A_r, θL_l, DL_l, y A_l) del registro de marcas naturales que recibe como parámetro para realizar la detección de puertas. Una puerta se detecta identificando una discontinuidad derecha (r) seguida de una discontinuidad izquierda (l).

La posición relativa de la puerta respecto al robot se obtiene con las ecuaciones 4.1 y 4.2. Un ejemplo de detección de puertas se muestra en la figura 4.6.

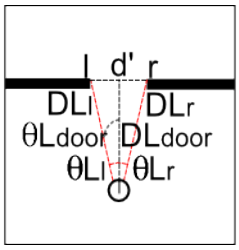


Figura 4.6: Detección de puertas.

$$\theta L_{door} = \frac{\theta L_r + \theta L_l}{2} \tag{4.1}$$

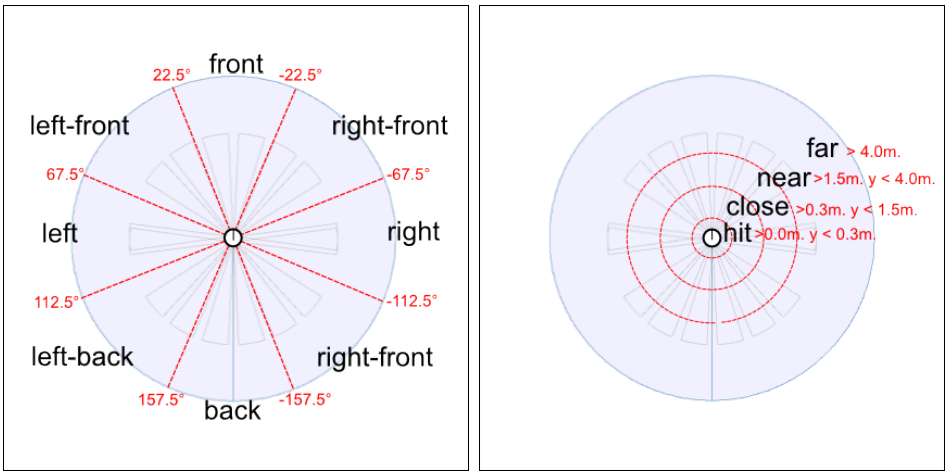
$$DL_{door} = \sqrt{DL_r^2 + DL_{\frac{d'}{2}}^2 - 2DL_r \frac{d'}{2} cos(\theta L_{door})} \tag{4.2}$$

$$d' = arcsen \frac{DL_r}{DL_l} \tag{4.3}$$

Los valores θL_{door} y DL_{door} corresponden a la posición (orientación y distancia), de la puerta detectada con respecto al robot. d' es la distancia entre r y l.

Una vez calculados los valores de orientación y distancia, el predicado obtiene las relaciones que existen entre el robot y las puertas detectadas. Las posibles relaciones de orientación que pueden existir entre el robot y las puertas son *right* (cuando la orientación de la puerta

se encuentra entre -67.5° y -112.5°), *left* (entre 67.5° y 112.5°), *front* (entre 22.5° y -22.5°), *back* (entre 157.5° y -157.5°), *right-back* (entre -112.5° y -157.5°), *right-front* (entre -22.5° y -67.5°), *left-back* (entre 112.5° y 157.5°) y *left-front* (entre 22.5° y 67.5°). Las posibles relaciones de distancia que pueden existir entre el robot y las puertas son *hit* (cuando la distancia a la puerta se encuentra entre 0.0$m.$ y 0.3$m.$), *close* (entre 0.3$m.$ y 1.5$m.$), *near* (entre 1.5$m.$ y 4.0$m.$) y *far* (> 4.0$m.$). La figura 4.7 muestra los rangos de valores de orientación y distancia utilizados para la generación de las descripciones relacionales.



(a) Rangos de valores de orientación. (b) Rangos de valores de distancia.

Figura 4.7: Rangos de valores para la generación de las descripciones relacionales.

Cada que se detecta una puerta, el predicado genera una lista con la descripción de la relación que existe entre dicha puerta y el robot así como los valores reales del ángulo de orientación y de la distancia que existen entre ellos. Por ejemplo, si θL_{door} = -162.2° y $DL_{door} = 2.33m.$, el predicado genera una lista con los elementos: [*right-back*, *near*, -162.2, 2.33]. Por cada puerta detectada se genera una de estas listas.

4.2.3 Predicado *walls_detected*

Utiliza los valores de los atributos tipo w (θL_{wall}, DL_{wall}, y $A_{wall} = L_{wall}$ o valor de la longitud de la pared) del registro de marcas naturales que recibe como parámetro para realizar la detección de paredes.

Los valores θL_{wall} y DL_{wall} corresponden a la posición (orientación y distancia), del punto central de las paredes detectadas con respecto al robot. El valor del atributo L_{wall} para las paredes corresponde a su longitud. Un ejemplo de detección de paredes se muestra en la figura 4.8.

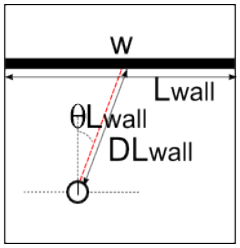


Figura 4.8: Detección de paredes.

Las relaciones de orientación o ángulo y distancia de las paredes se obtienen utilizando los mismos rangos de las puertas presentados anteriormente. Las posibles relaciones de longitud que pueden existir entre las paredes y el robot son *small* (cuando la longitud de la pared se encuentra entre $0.15m.$ y $1.5m.$), *medium* (entre $1.5m.$ y $4.0m.$) y *large* ($> 4.0m.$). La figura 4.9 muestra los rangos de valores de longitud utilizados para la generación de las descripciones relacionales.

Cuando se detecta una pared, el predicado genera una lista con la descripción de la relación que existe entre dicha pared y el robot así como los valores reales del ángulo de orientación y de la distancia que existen entre ellos. Por ejemplo, si $L_{wall} = 3.06m.$, $\theta L_{wall} = 36.9°$ y $DL_{wall} = 1.45m$, el predicado genera una lista con los elementos: [$medium$, $left$-

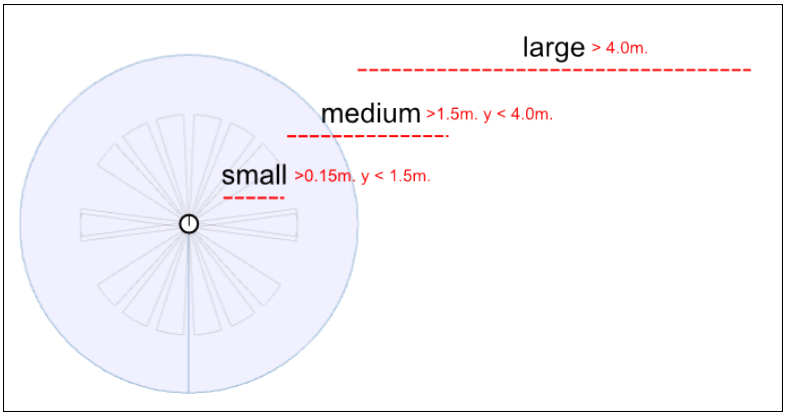


Figura 4.9: Rangos de valores de longitud de paredes para la generación de las descripciones relacionales.

front, *close*, 36.9, 1.45]. Por cada pared detectada se genera una de estas listas.

4.2.4 Predicado *corners_detected*

Utiliza los atributos tipo c (θL_{corner}, DL_{corner}, y A_{corner}) de la lista de descripciones de alto nivel que recibe como parámetro para realizar la detección de esquinas.

Los valores θL_{corner} y DL_{corner} corresponden a la posición (orientación y distancia), de las esquinas detectadas con respecto al robot. Las relaciones de orientación o ángulo y distancia de las esquinas se obtienen utilizando los mismos rangos de las puertas presentados anteriormente. Un ejemplo de detección de esquinas se muestra en la figura 4.10.

Cuando se detecta una esquina, el predicado genera una lista con la descripción de la relación que existe entre dicha esquina y el robot así como los valores reales del ángulo de orientación y de la distancia que existen entre ellos. Por ejemplo, si θL_{corner} =-71.6° y $DL_{corner} = 2.45m.$, el predicado genera una lista con los elementos: [*right*, *near*, -71.6, 2.45]. Por cada pared detectada se genera una de estas listas.

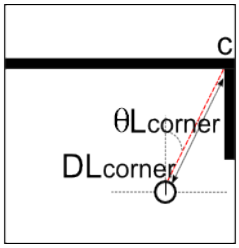


Figura 4.10: Detección de esquinas.

4.2.5 Predicado *obstacles_detected*

Utiliza los valores de los atributos tipo *o* ($\theta L_{obstacle}$, $DL_{obstacle}$, y $A_{obstacle}$) del registro de marcas naturales del ambiente que recibe como parámetro para realizar la detección de obstáculos.

Los valores $\theta L_{obstacle}$ y $DL_{obstacle}$ corresponden a la posición (orientación y distancia), de los obstáculos detectados con respecto al robot. Las relaciones de orientación o ángulo y distancia de los obstáculos se obtienen utilizando los mismos rangos de las puertas presentados anteriormente. Un ejemplo de detección de obstáculos se muestra en la figura 4.11.

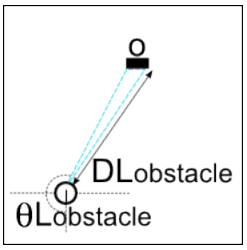


Figura 4.11: Detección de obstáculos.

Cuando se detecta un obstáculo, el predicado genera una lista con la descripción de la relación que existe entre dicho obstáculo y el robot así como los valores reales del ángulo de orientación y de la distancia que existen entre ellos. Por ejemplo, si $\theta L_{obstacle}$ =-21.2° y

$DL_{obstacle} = 4.3m.$, el predicado genera una lista con los elementos: [$front$, far, -21.2, 4.3]. Por cada obstáculo detectado se genera una de estas listas.

4.2.6 Predicado *goal_position*

Determina la orientación y distancia del robot respecto a la posición de destino o meta. Este predicado recibe como parámetros la posición del robot (x_r, y_r) y la posición del punto de destino (x_d, y_d). La posición (en coordenadas polares) del robot respecto a la meta se calcula con las ecuaciones 4.4 y 4.5.

$$\theta_{r-d} = arctan(\frac{(x_r - x_d)}{(y_r - y_d)}) \tag{4.4}$$

$$D_{r-d} = \sqrt{(x_r - x_d)^2 + (y_r - y_d)^2} \tag{4.5}$$

En donde θ_{r-d} y D_{r-d} representan, respectivamente, la diferencia relativa entre la posición (orientación y distancia) del robot y el destino. Las relaciones de orientación o ángulo y distancia de la posición de destino se obtienen utilizando los mismos rangos de las puertas presentados anteriormente.

4.2.7 Predicado *in_dest*

Verifica la distancia que existe entre la posición del robot y la posición del destino o meta. Si la distancia es menor a $0.5m.$ el predicado genera como resultado el valor *true*, caso contrario genera el valor *false*.

Los predicados anteriores indican si el robot se encuentra en una habitación, un pasillo o una intersección; indican si el robot está "observando" paredes, esquinas, puertas y obstáculos y cuales son sus características; y proporcionan una estimación de la orientación y distancia que existe hacia la meta o destino.

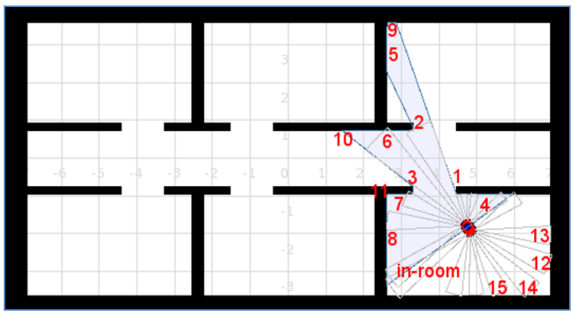
Análogos a los r-estados, las r-acciones son conjunciones de predicados de primer orden que se utilizan para caracterizar las diferentes acciones que puede ejecutar el robot en cada estado. Los predicados predefinidos que se utilizan para caracterizar las acciones del robot (generar las r-acciones) se presentan a continuación. Estos predicados reciben como parámetro los valores de velocidad y ángulo del robot los cuales se obtienen a través del odómetro del robot.

- *go*: Determina la acción de avance o retroceso que el robot realizó en el estado actual, sus posibles valores son $front$ (si la velocidad del robot $\geq 0.1 m./s.$), *back* (si la velocidad del robot $\leq$ -$0.1 m./s.$) y *nil* (si velocidad del robot se encuentra entre -$0.1 m./s.$ y $0.1 m./s.$). Este predicado genera una lista con la descripción de la acción de avance que ejecutó el robot así como los valores reales de velocidad de éste. Por ejemplo, si $speed = 0.52$ m./s., el predicado genera una lista con los elementos: [$front$, 0.52].

- *turn*: Determina la acción de giro que el robot realizó en el estado actual, sus posibles valores son: *slight-right* (si el ángulo del robot $\geq$ -45°), *right* (si el ángulo del robot <-45° y ≥-135°), $far\text{-}right$ (si el ángulo del robot < -135°), *slight-left* (si el ángulo del robot $\leq$ 45°), $left$ (si el ángulo del robot $\geq$ 45° y $\leq$ 135°) y $far\text{-}left$ (si el ángulo del robot $\geq$ 135°). Este predicado genera una lista con la descripción de la acción de giro que ejecutó robot así como los valores reales del ángulo en el cual está girando. Por ejemplo, si *angle* =-85°, el predicado genera una lista con los elementos: [$right$, -85].

Esta representación relacional de estados y acciones describen los elementos del ambiente que el robot está "observando" en algún instante de tiempo determinado, no depende de posiciones exactas de los elementos del ambiente como lo hacen las representaciones proposicionales, por lo tanto puede ser utilizada para transferir conocimiento entre diferentes dominios o ambientes interiores que contengan habitaciones, paredes planas, esquinas, pasillos y puertas como son los ambientes tipo hogar u oficina.

La figura 4.12 muestra un ejemplo de un par r-estado-r-acción generado a través de los anteriores predicados. Este par corresponde a los valores de las marcas naturales del ambiente y de la ubicación local o estancia actual del robot de la figura 4.5 mostrada previamente.



r-estado
location(in-room),
doors_detected([front, near, 0.1, 1.35]),
walls_detected([[right, close, medium, -35.7, 0.8], [front, far, small, -8.5, 4.6]...]),
corners_detected([[front, far, -14.5, 5.79], [front, near, 22.3, 2.3], [front-left, near, 31.6, 1.68]]),
obstacles_detected([[right-back, near, -170, 1.87], [back, near, 180, 42.5], [left-back, near, 150, 1.43]]),
goal_position([left-front, far]),
in_dest(false).

r-acción
go([front, 0.5]), **turn**([right, -60.0]).

Figura 4.12: Ejemplo de par r-estado-r-acción obtenido con la información de las marcas naturales del ambiente detectadas, la ubicación actual del robot y las lecturas de su odómetro.

Al utilizar estos predicados de primer orden, las representaciones de los elementos del ambiente son más abstractas, son comunes a varios estados, i.e., pueden ser re-utilizables y son más fáciles de comprender. Además, con representaciones relacionales como esta, es posible incluir o ampliar de forma sencilla, el conjunto de predicados para la caracterización de nuevos r-estados. Por ejemplo si se incluye el predicado *robot_gripper* con posibles valores como: *open*, *close*, *holding*, es posible caracterizar los diferentes estados de la(s) pinza(s) del robot.

Como se puede apreciar en la figura 4.12, algunos de los predicados, además de generar las correspondientes descripciones de r-estados y de r-acciones regresan valores numéricos. Los predicados que generan las descripciones del r-estado, como son el de detección de puertas,

paredes, esquinas y obstáculos, regresan los valores numéricos de orientación y distancia de cada uno de los elementos que detectan. Los predicados que generan las descripciones de la r-acción regresan los valores numéricos de velocidad y ángulo del odómetro del robot.

Los valores numéricos de orientación, distancia, velocidad y ángulo serán utilizados en la segunda fase del presente método para generar las políticas con acciones continuas a través de regresiones pesadas locales (LWR). Las descripciones de las relaciones de los r-estados y de las r-acciones se utilizarán en la primera fase del presente método para generar la política con acciones discretas a través de *Q-learning*.

4.3 TS-RRLCA primera fase

Cuando se quiere aprender una tarea o generar una política, utilizando por ejemplo aprendizaje por refuerzo, lo que interesa es intentar sólo acciones relevantes en cada estado. Es decir, en lugar de definir, de forma manual, un conjunto predefinido de acciones por estado, como en los modelos clásicos de aprendizaje por refuerzo, se utilizan trazas proporcionadas por el usuario para aprender sólo un pequeño subconjunto de r-acciones relevantes por cada r-estado. Este método, llamado clonación de comportamiento (Bratko et al. 1998), simplifica los procesos de aprendizaje al disminuir la cantidad de información que los algoritmos (por ejemplo, *Q-learning*) deben procesar.

4.3.1 Clonación de comportamiento para la inducción de r-estados y r-acciones

La primera fase de $TS\text{-}RRLCA$ comienza con un conjunto de ejemplos o trazas de la tarea que se quiere que el robot aprenda. Una traza (τ) es un registro de todas las lecturas de los sensores del robot (odómetro, láser y sonares) mientras desarrolla tal ejemplo de la tarea. Un registro de una traza se divide en observaciones o $frames$ ($\tau = \{f_{t=1}, f_{t=2}, ..., f_{t=n}\}$). Cada

$frame$ contiene los valores de todos los sensores del robot en un tiempo t determinado ($f_{t=k} = \{laser_1 = 2.25, laser_2 = 2.27, laser_3 = 2.29, ..., sonar_1 = 3.02, sonar_2 = 3.12, sonar_3 = 3.46, ..., velocidad = 0.48, angulo = 1.52\}$), es decir, cada $frame$ contiene los datos de bajo nivel de los sensores del robot (figura 4.13).

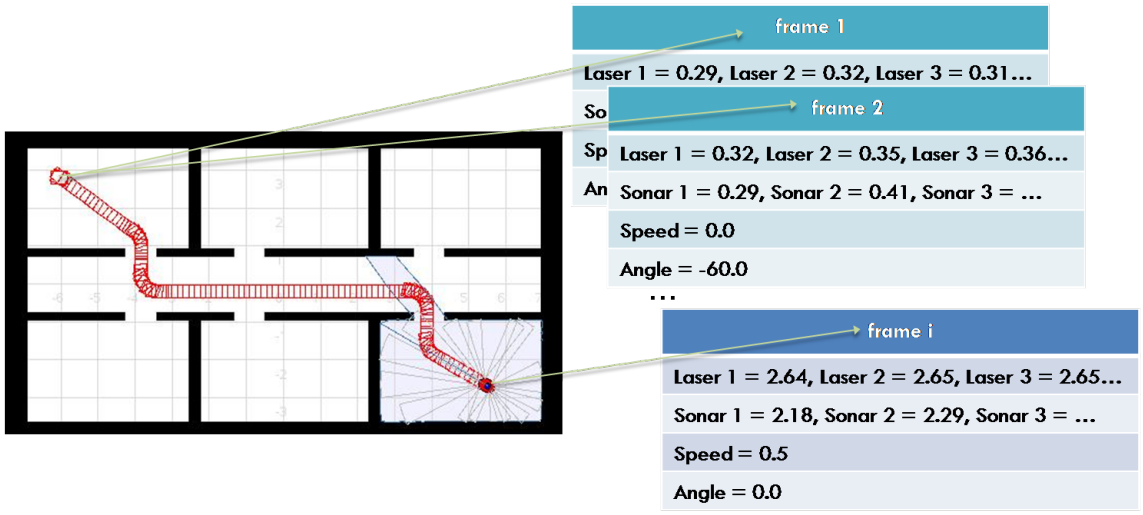


Figura 4.13: Ejemplo de traza y sus correspondientes $frames$.

Una vez generados diversos ejemplos o trazas ($\tau_1, \tau_2, ...\tau_m$) de la tarea que se quiere aprenda el robot; cada $frame$ de este conjunto de trazas, es transformado a sus correspondientes marcas naturales del ambiente y a su correspondiente ubicación local o estancia actual del robot. Estas marcas naturales y la ubicación del robot son enviadas como parámetro a los predicados de primer orden para obtener el correspondiente r-estado. La correspondiente r-acción se obtiene con los valores de velocidad y ángulo del $frame$. Así, por cada $frame$ de las trazas se obtiene un par r-estado-r-acción. La figura 4.14 muestra un ejemplo de la traza de $frames$, mostrada anteriormente, pero cada $frame$ ha sido convertido en su correspondiente par r-estado-r-acción.

Los pares r-estado-r-acción de cada una de las trazas son almacenados en una misma base de datos BD (figura 4.15).

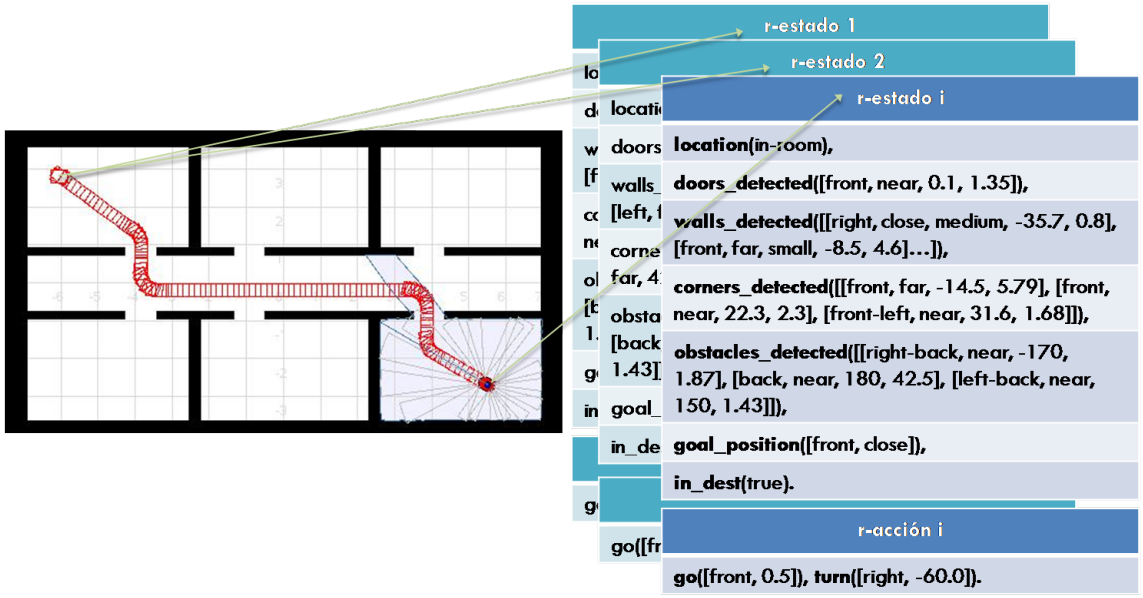


Figura 4.14: Ejemplo de traza y sus correspondientes pares r-estado-r-acción.

El algoritmo 4.1 muestra el pseudo-código del esquema de clonación de comportamiento descrito anteriormente, utilizado para convertir las trazas de *frames* en trazas de pares r-estado-r-acción.

Los pares r-estado-r-acción proporcionan la secuencia de acciones o trayectorias que el usuario realizó para completar determinada tarea. La finalidad es que el robot, a partir de estos pares, pueda generar un esquema de control (una política) que le permita imitar estas secuencias de acciones y con ello aprender a ejecutar la tarea que el usuario realizó en las trazas. Para ello, las descripciones, es decir, los valores discretos de los pares r-estado-r-acción, son proporcionados a un algoritmo de aprendizaje por refuerzo, llamado *rQ-learning*, para generar una política de control con acciones discretas. Los valores numéricos se utilizarán para transformar, a través de las regresiones pesadas locales, esta política con acciones discretas en una política con acciones continuas.

El esquema de clonación de comportamiento, presentado anteriormente, puede ser utilizado de forma incremental, en donde nuevas trazas pueden ser incorporadas en cualquier

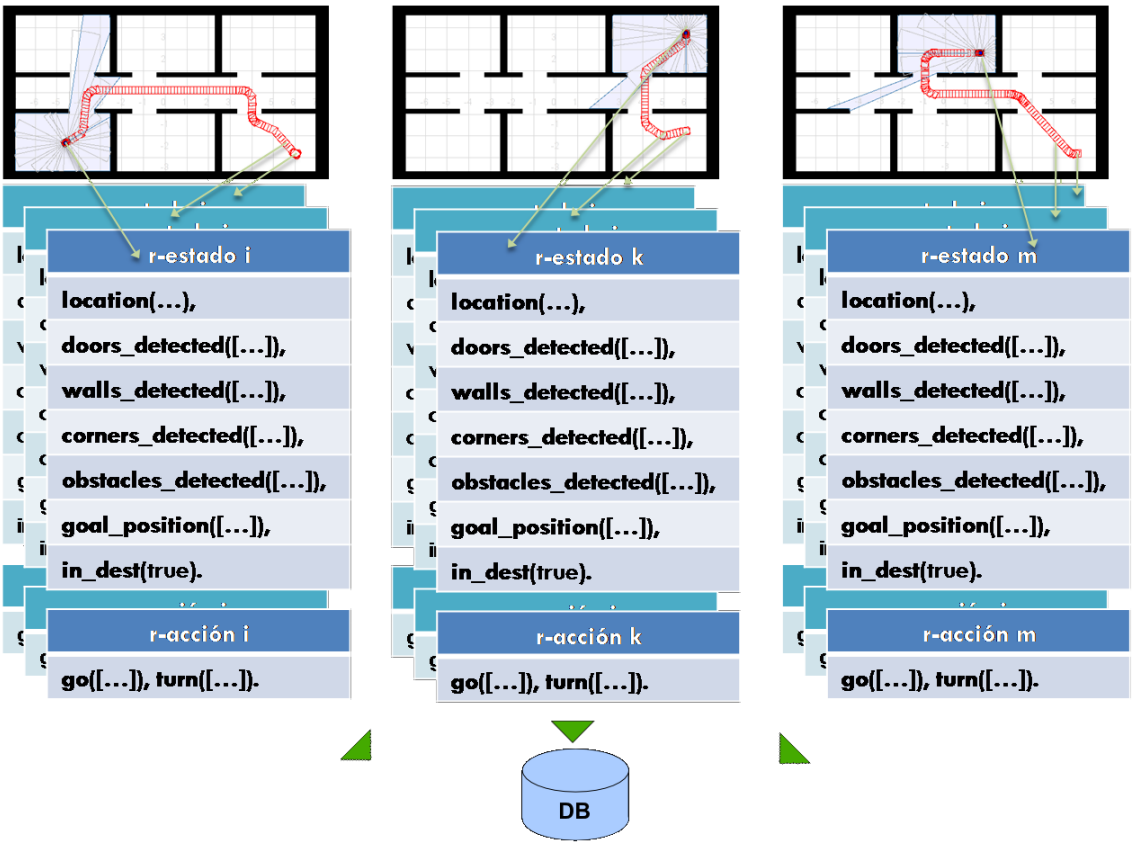


Figura 4.15: Trazas de pares r-estado-r-acción siendo almacenadas en la base de datos DB.

momento (posiblemente) incrementando el conjunto de r-estados y de r-acciones.

4.3.2 Aplicar RL al conjunto de r-estados y r-acciones para obtener una política de control relacional con acciones discretas

Después de obtener y almacenar los pares r-estado-r-acción del conjunto de trazas, se procede a la generación de las políticas de control a través de aprendizaje por refuerzo. Debido a que las trazas de las cuales se obtuvieron los pares r-estado-r-acción, pueden ser

Algoritmo 4.1 Algoritmo de clonación de comportamiento.

Entrada $\tau_1, \tau_2, ...\tau_n$: Conjunto de n trazas de la tarea que se quiere aprender.

Salida DB: Base de datos de pares estado-acción.

Para $i = 1$ **hasta** n **hacer**

$k \leftarrow$ número de $frames$ de la traza i

Para $j = 1$ **hasta** k **hacer**

Transformar el $frame_{ij}$ ($frame\ j$ de la traza i) en sus correspondientes marcas naturales del ambiente y en la correspondiente ubicación del robot.

Utilizar estas marcas naturales y la ubicación del robot para obtener el respectivo r-estado (a través de los predicados de primer orden).

Utilizar la velocidad y el ángulo del robot para obtener la respectiva r-acción.

$DB \leftarrow DB \cup$ {r-estado,r-acción}. *% Cada registro de DB se compone de un r-estado en conjunto con una r-acción*

Fin para

Fin para

proporcionadas por varias personas y a que en distintas trazas el robot pudo haber visitado los mismos lugares en el ambiente, es posible que se hayan realizado varias r-acciones en un mismo r-estado. Aprendizaje por refuerzo se utiliza para determinar qué r-acción es la adecuada para ser ejecutada en cada estado relacional.

Para comenzar con este proceso se coloca al robot en alguna posición inicial del ambiente (figura 4.16) y se obtienen los datos de bajo nivel de los sensores láser y sonares del robot.

Estos datos de bajo nivel se transforman en marcas naturales del ambiente y se obtiene la correspondiente ubicación del robot (figura 4.17).

Las marcas naturales y la ubicación se envían a los predicados de primer orden para que generen el conjunto de relaciones que se registran en el estado actual del robot, i.e., generen el r-estado actual del robot $r\text{-}estado_t$ (figura 4.18).

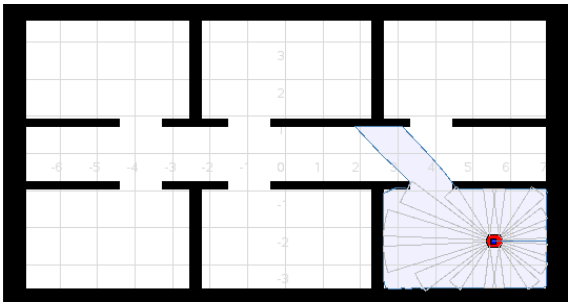


Figura 4.16: Robot en un estado inicial del ambiente.

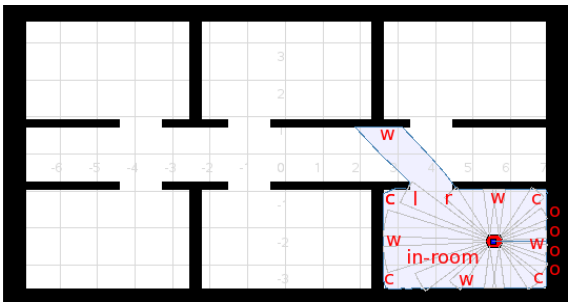
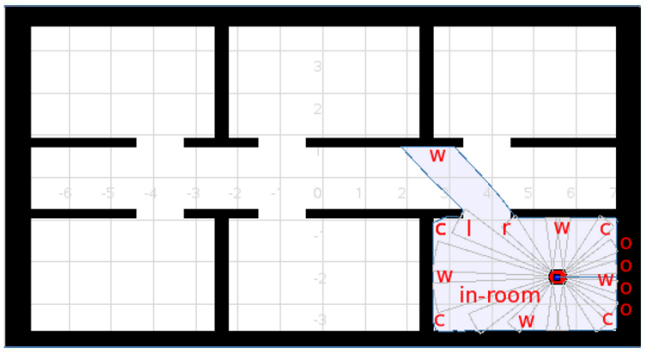


Figura 4.17: Marcas naturales y ubicación del robot correspondientes al estado inicial del ambiente.

Como se mostró anteriormente en la sección 4.2, cada par r-estado-r-acción contiene una parte nominal y una parte numérica. La parte nominal corresponde a la descripción del r-estado del robot, es decir, indica los elementos del ambiente que el robot percibió en un momento determinado, como puede ser una puerta al frente, una pared a la derecha y otra a la izquierda, etc. La parte numérica corresponde a las posiciones polares reales de esos elementos que el robot percibió como pueden ser, para la puerta: 2°, 2.36 m., para la pared derecha: -89.6°, 1.67 m. y para la pared izquierda: -197.18°, 3.04 m. Para aplicar aprendizaje por refuerzo a la base de datos de los pares r-estado-r-acción se utilizan sólo los valores de las descripciones nominales (marcados en color rojo en la figura 4.18). Los valores numéricos, por el momento, no serán utilizados sino hasta la segunda fase del presente método. Así que, una vez obtenido el r-estado actual del robot, se busca en la base de datos DB a todos aquellos



r-estado
location(in-room),
doors_detected([right-front, near, 0.52, 2.14]),
walls_detected([[right, close, medium, -81.4, 1.26], [front, near, medium, 6.4, 2.74],...]),
corners_detected([[right-back, close, 153.09, 1.17], [right-front, near, 52.6, 2.24], [left-front, near, -67.8, 2.1]...]),
obstacles_detected([[back, close, -170.0, 1.44]...]),
goal_position([front, close]),
in_dest(true).

Figura 4.18: Par r-estado-r-acción del robot correspondiente al estado inicial del ambiente.

registros que contengan la misma descripción, i.e., los mismos valores nominales que este r-estado actual. Los registros leídos de DB además de contener la misma descripción o valores nominales que el r-estado actual del robot, contienen las r-acciones que el usuario ejecutó en las trazas (figura 4.19).

El robot selecciona una de esas r-acciones que fueron leídas de DB siguiendo una política de selección de acciones como, por ejemplo, ϵ-*greedy* y posteriormente la ejecuta. Los valores que pueden tomar las descripciones de las r-acciones son:

- Para el predicado *go*: sus posibles valores son $front = 0.5m.$, $back = -0.5m.$ y $nil = 0.0m.$

- Para el predicado *turn*: sus posibles valores son: $slight\text{-}right$ = -45°, $right$ = -90°, $far\text{-}right$ = -135°, $slight\text{-}left$ = 45°, $left$ = 90°, $far\text{-}left$ = 135° y nil = 0°.

Estos valores de r-acciones corresponden al conjunto A de acciones discretas que se utilizarán en el presente trabajo de tesis para *rQ-learning*. Así que, si el robot elige, por ejemplo, ejecutar la r-acción del par m (mostrado en la figura 4.19): **go**(nil, 0.00), **turn**(right, -89.28),

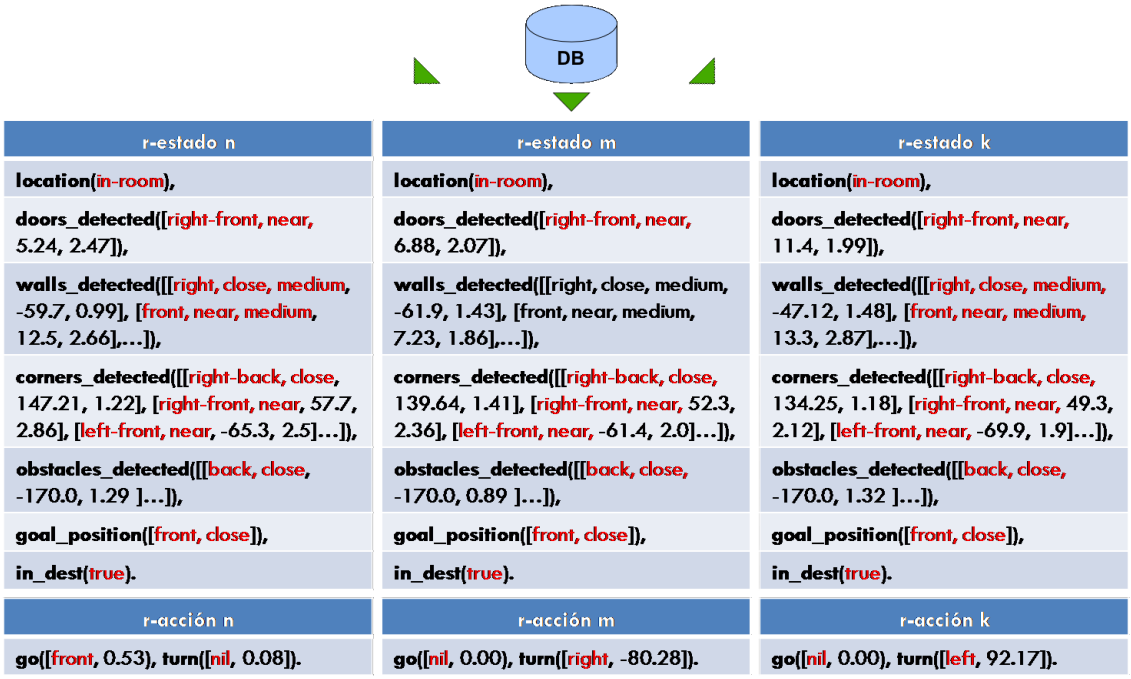


Figura 4.19: Pares r-estado-r-acción leídos de la base de datos DB y sus correspondientes valores nominales.

no ejecutará la acción de girar -89.28° como está en los valores de esa r-acción; lo que realmente hará es leer los valores nominales de la r-acción que en este caso son para **go** el valor de *nil* y para **turn** el valor de *right*. Estos valores nominales los buscará en el conjunto de acciones A que acabamos de definir y a cada valor nominal le asignará su correspondiente valor numérico. De acuerdo a los valores del conjunto A a la r-acción *nil* en el predicado **go** le corresponde un valor de velocidad de $0.0m$ y a la r-acción *right* del predicado **turn** le corresponde un ángulo de giro de -90°. Estos serán los valores reales de la r-acción que el robot ejecutará. Es decir, no se toman en cuenta los valores de la r-acción que se leen de DB, sólo se toma su descripción y a esta descripción se le asigna uno de los valores discretos de A previamente definidos.

Después de ejecutar la acción, el robot realiza una transición a un nuevo estado ($r\text{-}estado_{t+1}$). Una vez en el nuevo estado se actualizan los valores de la función Q y se comienza

nuevamente este proceso hasta llegar a un estado final. Posteriormente se realiza una nueva iteración hasta que los valores de la función Q converjan.

El algoritmo que lleva a cabo este proceso se presenta en el algoritmo 4.2 (Morales y Sammut 2004). Éste es una variante de *Q-learning* el cual fue presentado en la sección 2.1.1 de la presente tesis. Dado que los estados y las acciones se representan de forma relacional, es necesario modificar *Q-learning* para que pueda generar la política de control con estas representaciones relacionales en lugar de hacerlo con estados y acciones primitivos como los mostrados en la sección 2.1.1. Este algoritmo es muy similar a *Q-learning* excepto que los estados y las acciones están caracterizados de forma relacional.

Con este algoritmo se genera una política de control relacional con acciones discretas. Esta política indica qué r-acción se debe ejecutar en cada estado relacional; Q es la función que mapea un r-estado a una r-acción.

Hasta esta primera fase, con el enfoque utilizado se obtiene en pocas iteraciones, una política de control adecuada para realizar tareas a través de acciones discretas. Esta política, al ser relacional, permite transferir conocimiento entre dominios, es decir, puede ser aplicada a dominios o ambientes diferentes pero similares. En la sección de experimentos se mostrará cómo el presente método genera estas políticas relacionales con acciones discretas en pocas iteraciones. Además se mostrará cómo las políticas pueden ser utilizadas en ambientes diversos y son robustas a cambios en las metas u objetivos de las tareas.

El siguiente paso es transformar esta política con acciones discretas en una política con acciones continuas.

4.4 TS-RRLCA segunda fase

Las políticas generadas en la primera fase indican la acción que el robot debe ejecutar cuando se encuentra en determinado estado para completar cierta tarea. Sin embargo, las acciones sólo pueden tomar un conjunto finito de valores. Debido a esto, tareas que requieren

Algoritmo 4.2 Algoritmo rQ-learning.

Entrada DB: base de datos de pares r-estado-r-acción.

Salida Q: función que determina la política de toma de acciones del agente.

Inicializa $Q(S_t, A_t)$ arbitrariamente

Repite

$s_t \leftarrow$ valores de las lecturas de los sensores del robot.

Transformar s_t a sus correspondientes marcas naturales y ubicación del robot.

$S_t \leftarrow r\text{-}estado(s_t)$ % enviar las marcas naturales y la ubicación del robot a los predicados de primer orden para generar el r-estado correspondiente.

Para cada paso del episodio **hacer**

Buscar la descripción del r-estado S_t en la base de datos DB

Para cada registro en DB que contenga la descripción de r-estado de S_t **hacer**

Leer su correspondiente r-acción (A_t)

Fin para

Selecciona una acción A_t en S_t siguiendo alguna política de selección de acciones (e.g., ϵ-$greedy$)

Ejecuta acción A_t, observa r_{t+1}, s_{t+1}

Transformar s_{t+1} a sus correspondientes marcas naturales y ubicación del robot.

$S_{t+1} \leftarrow r\text{-}estado(s_{t+1})$ % enviar las marcas naturales y la ubicación del robot a los predicados de primer

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(r_{t+1} + \gamma max_{A_{t+1}} Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$

$S_t \leftarrow S_{t+1}$

Fin para

Hasta que s_t sea terminal

cierto nivel de precisión por parte del robot, como puede ser la transportación de objetos, la navegación a través de sitios muy saturados o el seguimiento de personas a través de

lugares muy concurridos, por lo general, no pueden ser llevadas a cabo correctamente. Con la finalidad de mejorar el desempeño de las tareas que lleva a cabo el robot, se transforman las políticas de control con acciones discretas a políticas con acciones continuas.

4.4.1 Regresión pesada local para la generación de políticas con acciones continuas

La idea de esta segunda fase es combinar los valores de las acciones discretas que proporciona la política generada en la fase previa, (**go** = {0,5m, -0,5m} y **turn** = {-90,0°, 90,0°}) con los valores de las r-acciones que ejecutó el usuario en las trazas y que fueron almacenadas en DB. De esta manera el robot ejecuta las acciones discretas que le dicta la política generada en la primera fase, pero los valores de las acciones se modifican a través de una regresión pesada local.

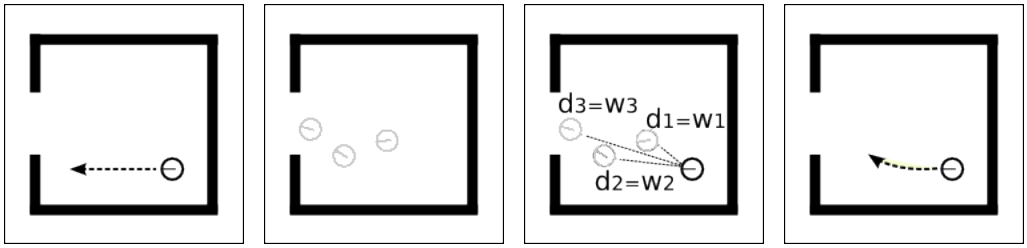
Esta regresión se realiza en cuatro partes:

1. Determinar el r-estado actual del robot y la correspondiente r-acción discreta que se debe ejecutar en este estado (figura 4.20(a)).

2. Obtener los valores de todas las r-acciones que el usuario realizó en este mismo r-estado actual cuando generó las trazas. Estos valores se encuentran almacenados en los registros de pares r-estado-r-acción de DB (figura 4.20(b)).

3. Asignar un peso o ponderación a los valores de esas r-acciones que realizó el usuario. Este valor de peso estará en función de la distancia que existe entre la posición del robot en el estado actual y la posición que tenía el robot en las trazas. Mientras más cercano, mayor será la influencia de los valores de sus acciones (figura 4.20(c)).

4. Combinar los valores de la r-acción discreta con los valores de esta ponderación para con ello generar una acción continua (figura 4.20(d)).



(a) r-estado actual y su correspondiente r-acción discreta

(b) Registros en DB que contienen la misma descripción del r-estado actual

(c) Cálculo de distancias y pesos

(d) Acción continua resultantes

Figura 4.20: Proceso de generación de acciones continuas.

A continuación se explica a detalle cada una de estas cuatro partes.

Determinar el r-estado actual del robot y la correspondiente r-acción

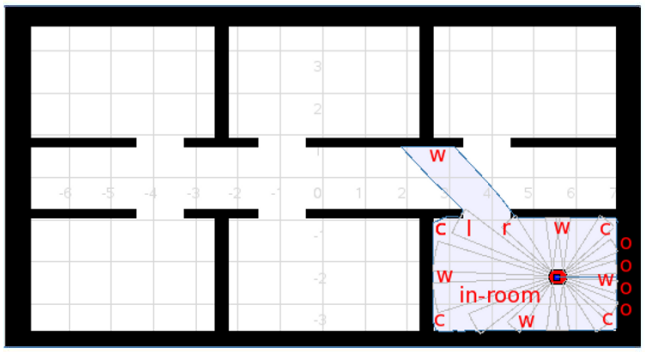
Al igual que en la primera fase, se coloca al robot en un punto inicial del ambiente y se obtienen los valores de sus sensores láser y sonares. Estos datos se transforman en marcas naturales y se obtiene la ubicación local del robot y, posteriormente, estas marcas naturales y la ubicación se envían a los predicados de primer orden para ser transformadas en un estado relacional $r\text{-}estado_t$ (figura 4.21).

Una vez identificado el r-estado actual del robot, la política de control generada en la fase previa, determina la acción a ser ejecutada ($\pi(r\text{-}estado_t)= r\text{-}acción_t$), la cual podría ser, por ejemplo, **go**(0,5), **turn**(0,00), figura 4.22.

Antes de ejecutar esta acción, el robot busca en DB todos los registros cuya descripción del r-estado sea igual a $r\text{-}estado_t$.

Obtener los valores de las r-acciones de las trazas del usuario

Los registros almacenados en DB contienen los pares r-estado-r-acción de las trazas. Como se vió en la sección 4.2 estos registros se conforman de dos partes, una parte contiene



r-estado
location(in-room),
doors_detected([right-front, near, 0.52, 2.14]),
walls_detected([[right, close, medium, -81.4, 1.26], [front, near, medium, 6.4, 2.74],...]),
corners_detected([[right-back, close, 153.09, 1.17], [right-front, near, 52.6, 2.24], [left-front, near, -67.8, 2.1]...]),
obstacles_detected([[back, close, -170.0, 1.44]...]),
goal_position([front, close]),
in_dest(true).

Figura 4.21: Par r-estado-r-acción del robot correspondiente al estado inicial del ambiente.

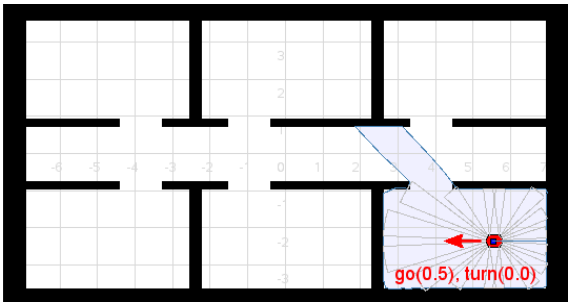


Figura 4.22: Robot en $r\text{-}estado_t$ y su correspondiente r-acción.

la descripción nominal del r-estado y de la r-acción, es decir, contiene un conjunto de valores no numéricos que describen los elementos que el robot "ve" en un momento determinado y la otra parte contiene valores numéricos que corresponden a la posición real de cada elemento. En la primera fase del método sólo se tomaron los valores nominales para generar la política de control a través de *rQ-learning* (sección 4.3.2), en esta segunda fase se utilizarán los valores numéricos.

De DB se leen todos aquellos registros que contengan la misma descripción nominal que

la del r-estado actual ($r\text{-}estado_t$). Dado que estos registros tienen la misma descripción que $r\text{-}estado_t$, implica que el usuario, durante las trazas, también pasó por este mismo r-estado en algún momento determinado.

Un ejemplo de pares r-estado-r-acción leídos de DB, que comparten la misma descripción de r-estado que $r\text{-}estado_t$, se muestra en la figura 4.23. En esta misma figura se muestran los valores numéricos de las posiciones reales de cada elemento así como de las acciones ejecutadas por el usuario en esa traza.

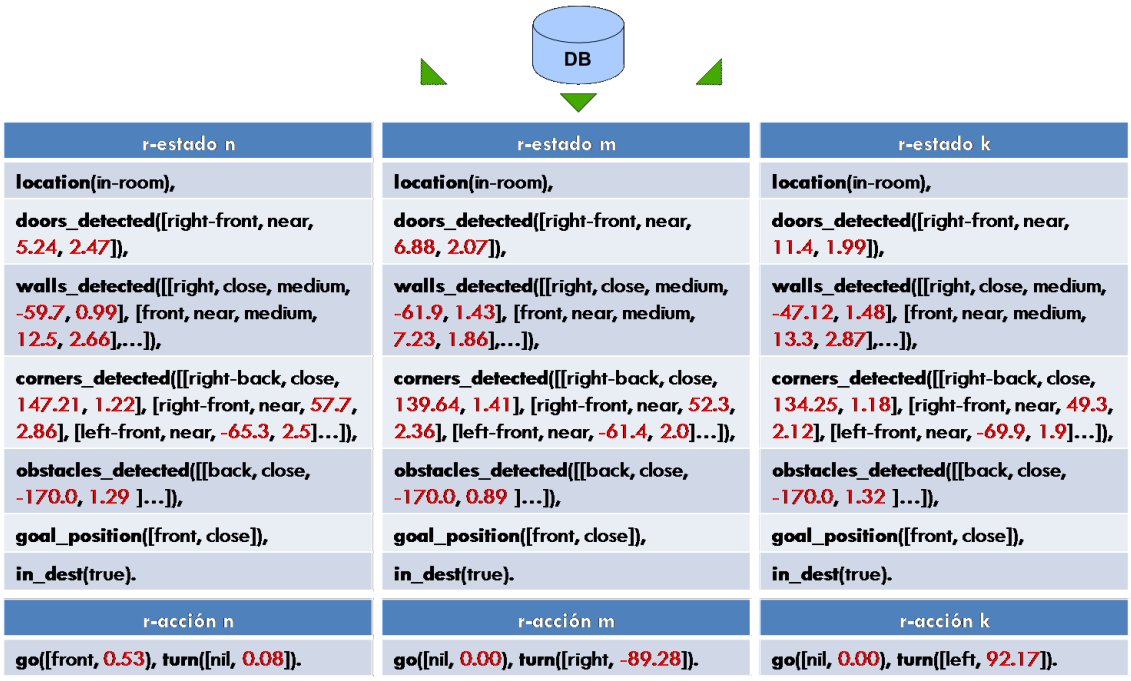


Figura 4.23: Pares r-estado-r-acción leídos de la base de datos DB y sus correspondientes valores numéricos.

Los valores numéricos de las r-acciones, leídos de DB, se utilizan para transformar los valores de la r-acción discreta que determinó la política en un valor continuo. Lo que se hace es utilizar los valores de la r-acciones que realizó el usuario durante las trazas para con ello modificar el comportamiento de la r-acción discreta.

Cálculo de distancias y pesos

El nivel de influencia que tendrá cada r-acción de las trazas depende de la distancia que existe entre el robot en el estado actual $r\text{-}estado_t$ y la posición en la cual se ejecutó dicha acción en las trazas. Dependiendo de este valor de distancia, se asigna un peso que determina qué tanto influyen los valores de las r-acciones de las trazas sobre los valores de la acción discreta.

Por ejemplo, en la figura 4.24 se muestra un ejemplo al robot en el r-estado actual $r\text{-}estado_t$ y también se muestran dos segmentos de trazas que comparten las mismas descripciones o valores nominales que el r-estado actual. Debido a que las posiciones por las cuales atravesó el robot en la traza a son más cercanas al robot en el estado actual que la traza b, los valores de las r-acciones de la traza a tendrán más peso que los de la traza b para transformar la r-acción discreta $r\text{-}acción_t$ en una r-acción continua.

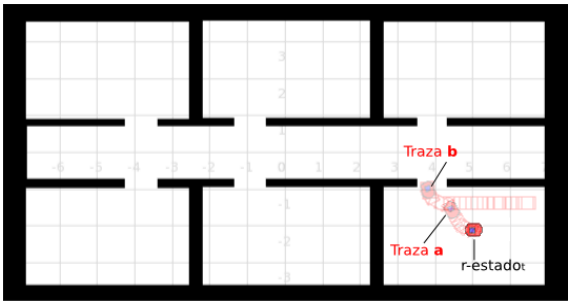


Figura 4.24: Trazas realizadas por el usuario las cuales comparten el mismo r-estado que $r\text{-}estado_t$.

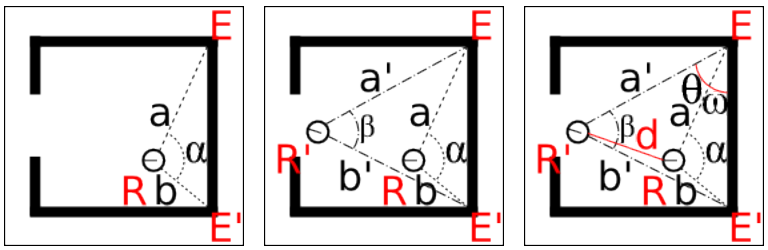
Debido a que no se almacenaron, en los registros de pares r-estado-r-acción, posiciones globales del robot, no es posible conocer de manera directa la distancia que existe entre el robot en el r-estado actual $r\text{-}estado_t$ (denotado con la letra R) y la posición o trayectoria seguida por el usuario en las trazas. Lo que si se conoce son los valores numéricos de orientación y distancia que registró el robot en las trazas (denotado con la letra R') y los elementos

del ambiente (elementos del r-estado marcados en color rojo en la figura 4.23). Estos valores se utilizan para calcular la posición entre el robot de las trazas R' y el robot en el estado actual R, a través de un proceso de triangulación. Este proceso de triangulación proporcionará la distancia relativa que existe entre el robot de las trazas con respecto al robot en el r-estado actual. La forma en la que se realiza la triangulación es a través de ecuaciones trigonométricas.

El proceso de triangulación se desarrolla como sigue. El robot R en el r-estado actual sensa su ambiente y detecta los elementos E y E' (figura 4.25(a)), los cuales pueden ser alguna esquina, una pared, una puerta, etc. Cada elemento tiene una distancia relativa (a y b) y un ángulo con respecto a R. Los ángulos no se utilizan de forma directa en este proceso de triangulación, lo que se utiliza es la diferencia absoluta entre estos ángulos (α).

Para este momento el robot ya ha leído de DB todos los registros que tienen la misma descripción del r-estado actual. Los valores numéricos de los ángulos y distancias de estos registros corresponden a las distancias relativas (a' y b') del robot de las trazas R' a los mismos elementos E y E' y al correspondiente ángulo β (figura 4.25(b)).

Para conocer la distancia (d) entre R y R' a través de este proceso de triangulación, se utilizan las ecuaciones 4.6, 4.7, 4.8 y 4.9.



(a) Robot R en el estado actual y sus correspondientes elementos identificados.

(b) Robot R' de las trazas, que tiene la misma descripción de r-estado que R.

(c) Elementos a ser calculados para conocer d.

Figura 4.25: Proceso de triangulación.

$$\overline{EE'} = \sqrt{a^2 + b^2 - 2ab\ cos(\alpha)} : \ Distancia\ entre\ E\ y\ E'. \tag{4.6}$$

$$\theta + \omega = arcsen(a'/\overline{EE'}) : \ Angulo\ entre\ a'\ y\ \overline{EE'}. \tag{4.7}$$

$$\omega = arcsen(a/\overline{EE'}) : \ Angulo\ entre\ a\ y\ \overline{EE'}. \tag{4.8}$$

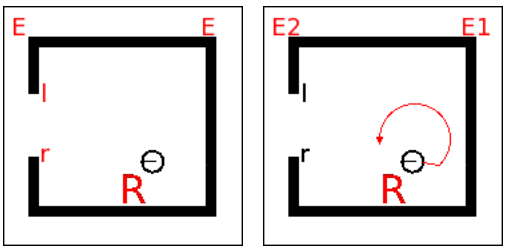
$$d = \sqrt{a^2 + a'^2 - 2aa'\ cos(\theta)} : \ Distancia\ entre\ R\ y\ R'. \tag{4.9}$$

Una vez que se ha calculado este valor de distancia d, se utiliza un $Kernel$, por ejemplo el $Kernel$ Gaussiano, para conocer su respectivo valor de peso w:

$$w(d) = exp(-d^2) \tag{4.10}$$

Siempre que se presenta el caso de que existan más de dos elementos para realizar esta triangulación, se toman los primeros dos elementos encontrados en las descripciones del r-estado. Por ejemplo, si el robot hubiera detectado cuatro elementos (figura 4.26(a)), la

triangulación se realiza con los primeros dos elementos detectados (figura 4.26(b)), esto, recordando que el proceso de identificación de elementos del ambiente se realiza a contrasentido de las manecillas del reloj, comenzando en la parte trasera central del robot (como se vio en la sección 4.1).



(a) Robot identificando elementos del ambiente.

(b) Robot seleccionando elementos para triangulación.

Figura 4.26: Proceso de selección de elementos del ambiente para la triangulación de su posición.

Combinar los valores de la r-acción discreta con los valores de las trazas

El valor de peso w se debe multiplicar por el valor numérico de velocidad y de ángulo de la r-acción de R', es decir, por los valores de la r-acción leídos de DB. Los valores resultantes de esta multiplicación se suman al valor de la r-acción discreta de R dando un valor de acción continuo.

Este proceso de triangulación y de asignación de pesos se aplica a cada uno de los registros leídos de DB que comparten la misma descripción de r-estado que R. Por ejemplo, si los registros que se leyeron de DB son los pares r-estado-r-acción denotados con las letras n, m y k que se mostraron en la figura 4.23, el proceso de triangulación y de asignación de pesos se debe hacer tres veces, una para cada registro leído de BD. Este proceso de cuatro pasos se aplica cada vez que el robot R llega a un nuevo estado.

El algoritmo que lleva a cabo el proceso de generación de acciones continuas se muestra en el algoritmo 4.3.

En el presente capítulo se expuso el método propuesto para generar políticas relacionales con acciones continuas a través de aprendizaje por refuerzo. El método se divide en dos fases, en la primera fase se genera una política de control relacional con acciones discretas a través de un algoritmo de aprendizaje por refuerzo llamado *rQ-learning*. Este algoritmo es una variación de *Q-learning* en donde los estados y las acciones no se representan de forma proposicional sino de forma relacional permitiendo que las políticas generadas a través de este esquema puedan ser re-utilizadas o transferidas entre dominios similares. La representación relacional de estados y acciones se realizó a través de predicados de primer orden que describen los elementos que el robot "observa" en un tiempo determinado y que describen la correspondiente acción que el robot ejecutó en este momento. En esta misma fase se utilizó clonación de comportamiento se logra acelerar el proceso de generación de las políticas de *rQ-learning*. En la segunda fase del método propuesto se utilizó un esquema de regresión pesada local para combinar la información que proporciona clonación de comportamiento en conjunto con los valores de las acciones de la política con acciones discretas generada por *rQ-learning*, para convertir tales acciones discretas en continuas.

El método propuesto contiene los siguientes aspectos novedosos:

- Se hace una transformación de los datos de bajo nivel de los sensores del robot a una representación basada en la ubicación del robot respecto a paredes, esquinas, puertas, obstáculos, etc., la cual permite: (i) un mayor entendimiento de las descripciones de estados debido a que las representaciones son más naturales o comprensibles al ser descritas en un lenguaje más cercano al lenguaje natural a través de predicados de primer orden, (ii) una reducción del espacio de estados, (iii) aprendizaje rápido de políticas debido al nivel de abstracción de la información y, (iv) capacidad de transferencia a otros ambientes tipo hogar u oficina con habitaciones, puertas, paredes, esquinas, etc.

Algoritmo 4.3 Algoritmo de generación de acciones continuas.

Entrada π: política relacional con acciones discretas.

DB: base de datos de pares r-estado-r-acción.

Salida Acciones continuas del robot.

Repite

$s_t \leftarrow$ valores de las lecturas de los sensores del robot.

Transformar s_t a sus correspondientes marcas naturales y ubicación del robot.

$S_t \leftarrow r\text{-}estado(s_t)$ % enviar las marcas naturales y la ubicación del robot a los predicados de primer orden para generar el r-estado correspondiente.

$A_t \leftarrow \pi(\text{r-estado}_t)$. % La política generada en la fase previa determina la acción discreta a ser ejecutada.

Buscar en DB los registros que contengan la descripción del r-estado S_t.

Para cada registro en DB que contenga la descripción de r-estado de S_t **hacer**

Leer los valores numéricos de orientación y distancia de los elementos del r-estado (puertas, paredes, esquinas, obstáculos).

Realizar la triangulación con estos valores, para que la posición del robot de las trazas (R') sea conocida por el robot en el r-estado actual $r\text{-}estado_t$.

Obtener la distancia d entre el robot en el r-estado actual y robot de las trazas.

Asignar un peso w utilizando algún *kernel* al valor de distancia d.

Multiplicar el valor del peso w por los valores numéricos de velocidad y ángulo de la r-acción.

Sumar a A_t el valor de la multiplicación anterior.

Fin para

Mientras no exista un cambio de estado **hacer**

Continuar ejecutando A_t

Fin Mientras

Hasta que S_t sea terminal

- Se consigue generar rápidamente una política con acciones discretas y se transforma a una política con acciones continuas que permite: (i) generarse en tiempo real y como veremos más adelante, producir trayectorias más cortas, mas suaves y con menos errores que las políticas con acciones discretas.

- El usuario proporciona trazas con lo cual: (i) se reduce el esfuerzo de programación de tareas al sólo tener que generar ejemplos de tareas y, (ii) se simplifica el proceso de aprendizaje por refuerzo al reducir el espacio de acciones consideradas en cada estado, haciendo mucho más rápida la generación de la política de control, lo cual se verá a detalle más adelante.

En el siguiente capítulo se exponen los experimentos realizados y los resultados obtenidos al utilizar el presente método para generar políticas de control de navegación y de seguimiento para un robot simulado y uno real.

5

Experimentos

El presente capítulo muestra los experimentos y resultados obtenidos luego de utilizar el método propuesto para la generación de políticas relacionales con acciones continuas. Con este método se generaron dos políticas, una para ejecutar tareas de navegación y otra para ejecutar tareas de seguimiento con robots móviles de servicio. Se muestra como las políticas generadas en un ambiente determinado pueden ser re-utilizadas en diferentes ambientes de tipo hogar y oficina. Se realiza una comparación entre el método propuesto con *Q-learning* y con aprendizaje por refuerzo relacional (Dzeroski et al. 1998). Esta comparación se realiza en términos de:

- Número de iteraciones que toma a cada método generar las políticas de control de navegación y de seguimiento.
- Tiempo de ejecución de las tareas.
- Error o desviación con respecto a esas mismas tareas cuando son ejecutadas por humanos.
- Espacio libre utilizado durante la ejecución de las tareas.

5.1 Características de los robots

El robot utilizado en las simulaciones en *player/stage* (R. et al. 2003) es un *pioneer2DX*. El robot real utilizado en estos experimentos es un robot *GuiaBot*[1] de *ActivMedia* con una base *PatrolBot*. En la figura 5.1 se muestran imágenes de este robot.

(a) Robot vista frontal. (b) Robot vista posterior. (c) Robot vista lateral.

Figura 5.1: Vistas del robot *Guiabot* de *ActivMedia*.

Ambos robots están dotados de un sensor láser que proporciona 180 lecturas correspondientes a los 180° frontales del robot (de $-90°$ a 90°) y 4 sonares ubicados en la parte posterior del robot (colocados a $-170°$, $-150°$, 150° y 170°). El rango de los sensores láser es de 8,0m. y para los sonares el rango es de 6,0m.

5.2 Generación de políticas

Las políticas que se generaron en el presente trabajo fueron: una política para realizar tareas de navegación a diferentes puntos dentro del ambiente del robot (política de nave-

[1] http://www.mobilerobots.com/PatrolBot.html

gación) y una política para realizar tareas de seguimiento de un objeto, en este caso, un segundo robot (política de seguimiento).

El proceso de entrenamiento o generación de políticas se llevó a cabo en simulación en *player/stage* y únicamente en el ambiente cuyo mapa se muestra en la figura 5.2.

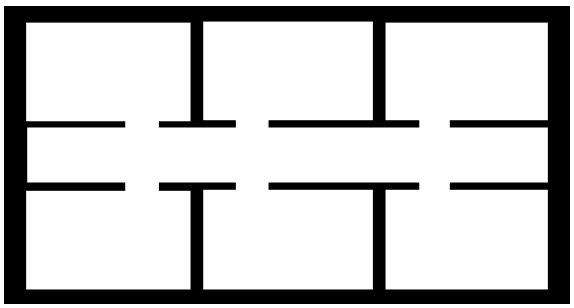


Figura 5.2: Mapa simulado donde se generaron las políticas de navegación y seguimiento, tamaño 15,0$m.$ × 9,0$m.$ (mapa 1)

En este mapa se realizaron las trazas que contienen los ejemplos de las tareas a aprender. En total se generaron dos conjuntos de trazas, uno contiene 20 trazas en las cuales sólo se realizaron tareas de navegación y otro conjunto contiene 20 trazas en las cuales sólo se realizaron tareas de seguimiento. Estas trazas fueron generadas en una simulación de *player/stage* en el mapa de la figura 5.2. Para generar estas trazas el usuario controlaba al robot por medio de un *joystick*. Para las trazas de navegación la posición de destino o meta de la tarea era indicada por el usuario y posteriormente controlaba al robot hasta que llegara a su destino. Para las trazas de seguimiento, la posición del robot a seguir se obtuvo a través de láser. Se supuso que el robot a seguir tenía un ancho determinado. La finalidad del sensor láser fue detectar algún objeto con ese ancho en particular y dirigirse hacia él. En las trazas de seguimiento el usuario controlaba al robot para que se acercara al robot que era perseguido.

Cada conjunto de trazas fue almacenado en una base de datos de trazas. *Player/stage*

proporciona la opción de simular los sensores láser y sonares del robot así como de realizar lecturas de estos sensores. Constantemente se leían estos sensores y sus lecturas se almacenaban en registros en una base de datos de trazas. Cada registro de la base de datos de trazas corresponde a un $frame$ o lectura de los datos de bajo nivel de los sensores láser y sonares del robot así como de su velocidad y ángulo obtenido del odómetro. Un ejemplo de traza de una tarea de navegación se muestra en la figura 5.3. Mientras la tarea no terminaba, es decir, mientras el robot no llegaba al punto de destino, constantemente se almacenaban registros de $frames$ o lecturas de los sensores.

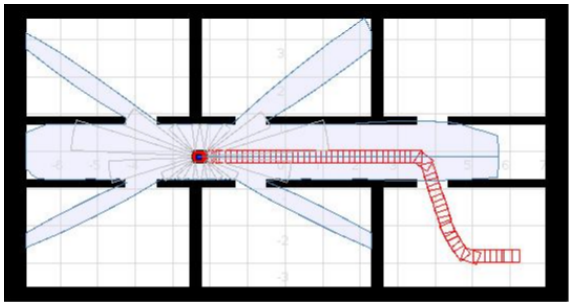


Figura 5.3: Ejemplo de una traza para el aprendizaje de tareas de navegación.

Un ejemplo de traza de una tarea de seguimiento se muestra en la figura 5.4. Mientras el robot no se encontrara a una distancia menor a $0.5m.$ del robot que era perseguido, constantemente se almacenaban registros de $frames$ o lecturas de los sensores.

A cada registro de cada base de datos de trazas se le aplicó el método para producir sus correspondientes marcas naturales del ambiente y la correspondiente ubicación local o estancia del robot. Estos datos fueron enviados a los predicados de primer orden para obtener los correspondientes pares r-estado-r-acción los cuales se almacenaron en dos base de datos DB, una que contiene los pares r-estado-r-acción de las trazas de navegación (DB_{nav}) y otra que contiene los pares r-estado-r-acción de las trazas de seguimiento (DB_{seg}).

Del conjunto de trazas de navegación se obtuvieron $7,346$ combinaciones de pares r-

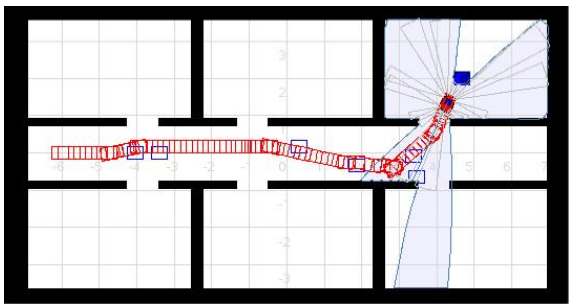


Figura 5.4: Ejemplo de una traza para el aprendizaje de tareas de seguimiento.

estado-r-acción y del conjunto de trazas de tareas de seguimiento se obtuvieron $10,018$, esto, contando los valores numéricos de los pares. Por cada descripción de un par r-estado-r-acción se obtuvo un promedio de 8 registros, es decir, de descripciones de pares r-estado-r-acción repetidas pero que contienen diferentes valores numéricos los cuales se utilizan para la regresión pesada local. Entonces en total, sin tomar en cuenta los registros con descripciones repetidas, se obtuvieron 934 pares r-estado-r-acción de las trazas de navegación y $1,406$ pares de las trazas de seguimiento de un total de $51,840$ combinaciones posibles[2] de pares r-estado-r-acción.

Con los pares r-estado-r-acción de las trazas de navegación, almacenados en DB_{nav}, se generó la correspondiente política de navegación relacional con acciones discretas haciendo uso del algoritmo de *rQ-learning*. Con los pares de las trazas de seguimiento, almacenados en DB_{seg}, se generó la correspondiente política de seguimiento relacional, de igual forma, a través de *rQ-learning*.

A continuación se muestran las curvas de aprendizaje para las políticas de navegación (figura 5.7) y de seguimiento (figura 5.10). En estas curvas de aprendizaje se compara el número de iteraciones que toma al método propuesto (*TS-RRLCA*) generar las políticas

[2]Aunque no es factible que se puedan llegar a conocer todas las combinaciones posibles de pares r-estado-r-acción se pone el total de combinaciones de los valores de los predicados utilizados para las descripciones de los r-estados y de las r-acciones.

de control con el número de iteraciones que le toma a aprendizaje por refuerzo (RL) y a aprendizaje por refuerzo relacional (RRL).

Para generar la política de control de navegación con RL (Watkins 1989), el mapa de entrenamiento, mostrado anteriormente en la figura 5.2, se dividió en estados de tamaño fijo de $25.0cms^2$ con lo cual se tiene un total de $2,160$ estados (figura 5.5).

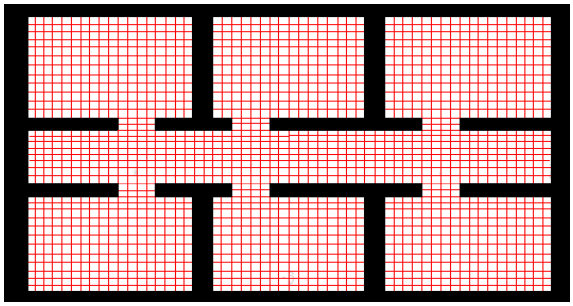


Figura 5.5: Mapa simulado segmentado en sus correspondientes estados.

El número de posibles acciones del robot por cada estado es de 8 las cuales corresponden a las mismas 8 acciones relacionales presentadas en la sección 4.2:

1. $front$: El robot avanza $25.0cms$.
2. *back*: El robot retrocede $25.0cms$.
3. *slight-right*: El robot gira -45°.
4. *right*: El robot gira -90°.
5. $far\text{-}right$: El robot gira -135°.
6. $slight\text{-}left$: El robot gira 45°.
7. $left$: El robot gira 90°.
8. $far\text{-}left$: El robot gira 135°.

Con lo que se tiene un total de $17,280$ combinaciones posibles de pares estado-acción. La política de control se generó utilizando el algoritmo presentado en la sección 2.1.1.

Para generar la política de control de navegación con *RRL* los estados fueron descritos de forma relacional haciendo uso de la representación presentada anteriormente. Al utilizar esta representación relacional no se depende de posiciones exactas o de tamaños fijos para la descripción de los estados dando como resultado estados más abstractos, i.e., que abarcan espacios más grandes del ambiente que los estados proposicionales de *RL* y que dependen de la proximidad del robot respecto a los objetos del ambiente (figura 5.6). Para *RRL* se utilizaron las descripciones de los r-estados generados durante las trazas de las tareas de navegación y que fueron almacenados en la base de datos DB_{nav}, pero no se toman los valores de las r-acciones. Así, por cada r-estado el robot deberá elegir entre todas las posibles r-acciones disponibles (descritas en la sección 4.2 del presente trabajo). En total para *RRL* se tienen 655 r-estados con 8 posibles r-acciones por cada r-estado, lo que da un total de $5,240$ posibles combinaciones de pares r-estado-r-acción.

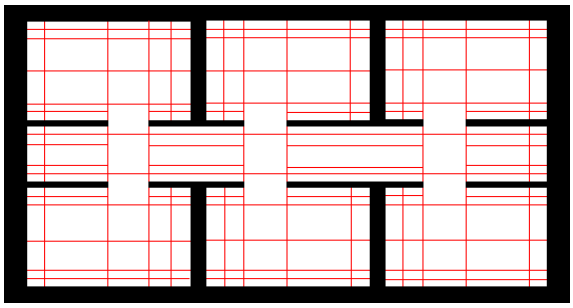


Figura 5.6: Mapa simulado segmentado en sus correspondientes estados relacionales.

En el método propuesto (*TS-RRLCA*), al igual que con *RRL* se utilizaron las descripciones de los r-estados de las trazas pero en esta ocasión en cada r-estado que visitaba el robot sólo se elegía entre las r-acciones que el usuario ejecutó en los ejemplos de las trazas en lugar de elegir entre todas las posibles r-acciones, así que, como se mencionó anteriormente,

el total de pares r-estado-r-acción con el método propuesto es de 934.

Para generar las políticas, los valores Q de *Q-learning* y de *rQ-learning* fueron inicializados en -1, ϵ en 0.1, γ en 0.9 y α en 0.1 (Sutton y Barto 1998). Un valor de refuerzo positivo de *+100* se otorga cuando se alcanza la meta, es decir, cuando la diferencia entre la distancia del robot al punto de destino es menor a 0.5 m. Un refuerzo negativo de *-5* se da cuando el robot choca con algún objeto, es decir, cuando algún elemento tiene, como descripción de distancia, el valor de *hit* (para ver los posibles valores de distancia referirse a la sección 4.2). Un valor de *-1* se da en cualquier otro caso.

Como se muestra en la figura 5.7, con el método propuesto los valores Q de *rQ-learning* convergen rápidamente (a su máximo valor posible: 100) y no presentan importantes mejoras en el desempeño de las políticas de navegación generadas después de aproximadamente $3,000$ iteraciones. En una computadora $PentiumDualCore$ a 2.2 Ghz y 3 Gb en RAM, a nuestro método le tomó un tiempo de 28 min. en llegar a las $3,000$ iteraciones para generar la política de navegación. Aprendizaje por refuerzo clásico (RL) requirió de $27,000$ iteraciones en un tiempo de 6 hrs. 51 min. Además de este tiempo requerido, la política de aprendizaje por refuerzo clásico sólo sirve para que el robot vaya hacia un único punto de destino o estado final (figura 5.8) ya que en este enfoque no se utilizan representaciones relacionales y las representaciones proposicionales, utilizadas por RL, no permiten cambios o variaciones en las tareas. Aprendizaje por refuerzo relacional (RRL) requirió de $13,000$ iteraciones para generar su política de control las cuales le tomaron un tiempo de 2 hrs. 16 min.

En la figura 5.9(a) se muestra un ejemplo de una tarea de navegación realizada por el robot en simulación utilizando la política generada por aprendizaje por refuerzo y en la figura 5.9(b) se presenta esa misma tarea pero ejecutada haciendo uso del método propuesto $TS\text{-}RRLCA$. La política de control de aprendizaje por refuerzo genera una trayectoria de navegación óptima (Sutton y Barto 1998) que es la que el robot sigue para dirijirse a su punto de destino. Sin embargo, la política generada con el método propuesto puede ser utilizada para que el robot se dirija de distintos puntos de origen a distintos puntos de destino debido

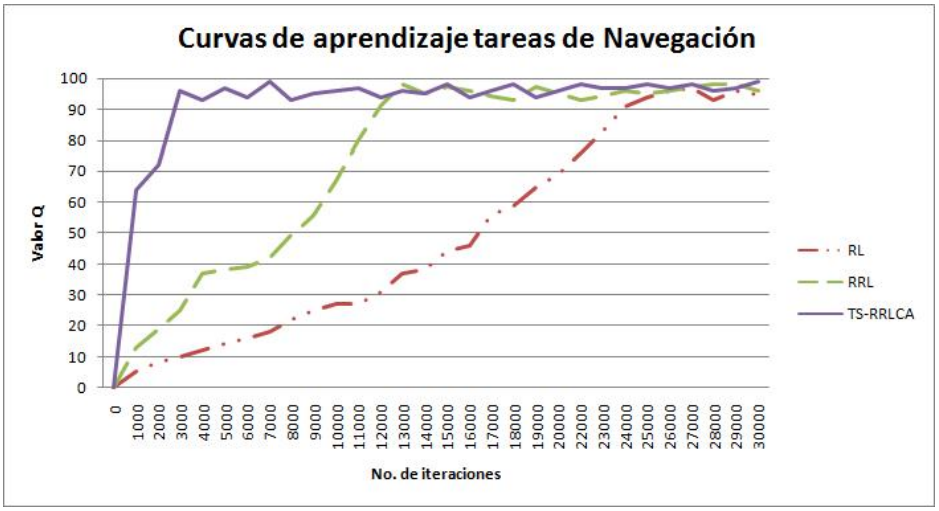


Figura 5.7: Curvas de aprendizaje para las políticas de navegación.

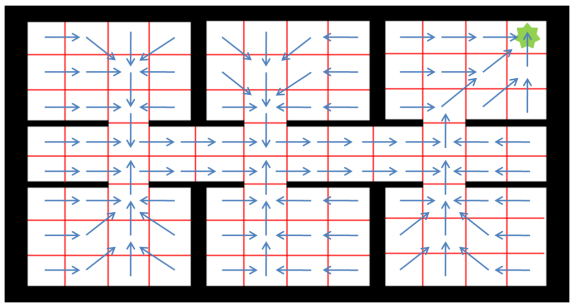
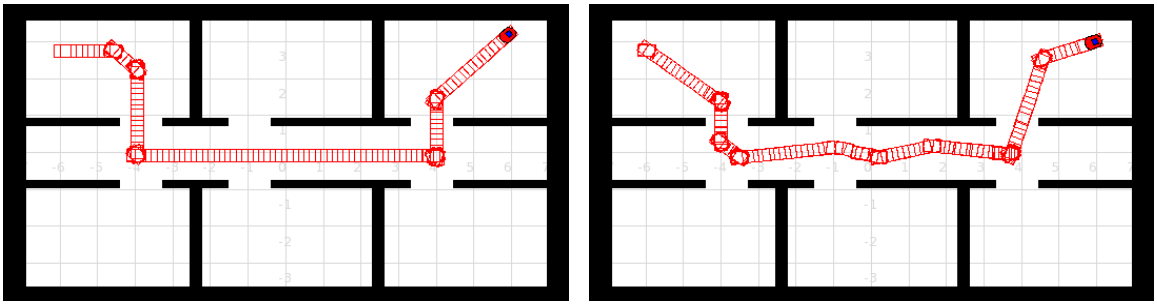


Figura 5.8: Política de navegación generada por aprendizaje por refuerzo con su correspondiente estado final

a que la representación relacional utilizada no depende de estados de tamaño fijo ni de posiciones exactas de los elementos del ambiente como es el caso de las políticas generadas con aprendizaje por refuerzo. Para que el robot se dirija a distintos puntos dentro de su ambiente, a través de aprendizaje por refuerzo clásico, sería necesario desarrollar diversas políticas de control y el método propuesto sólo emplea una. Además, el tiempo de generación de las políticas de aprendizaje por refuerzo es mucho mayor que el tiempo requerido por el

método propuesto.



(a) Tarea de navegación ejecutada por el robot móvil utilizando la política generada por aprendizaje por refuerzo.

(b) Tarea de navegación ejecutada por el robot móvil utilizando la política generada por *TS-RRLCA*.

Figura 5.9: Ejemplos de tareas de navegación ejecutadas con políticas generadas por aprendizaje por refuerzo y por *TS-RRLCA*.

Generar políticas de seguimiento a través de aprendizaje por refuerzo clásico (*RL*) no es posible de forma directa ya que se necesita de una descripción de estados y acciones que permita establecer relaciones entre el robot en el estado actual y el robot u objeto a ser seguido. Para generar este tipo de políticas se utilizan representaciones relacionales y es por tal motivo que surgió aprendizaje por refuerzo relacional *RRL* (Dzeroski et al. 1998).

Para *RRL* se utilizaron las descripciones de los r-estados almacenados en DB_{seg} y nuevamente no se tomaron en cuenta los valores de las r-acciones sino que se elegía entre todas las posibles r-acciones disponibles (descritas en la sección 4.2 del presente trabajo). El total de pares r-estado-r-acción fue de $3,149$.

Para *TS-RRLCA* si se utilizaron las r-acciones almacenadas en esta base de datos así que el robot sólo elegía una r-acción de un subconjunto del total de posibles r-acciones. El total de pares r-estado-r-acción, como ya se mencionó anteriormente, fue de $1,406$.

Para las políticas de seguimiento al método propuesto le tomo $4,000$ iteraciones para que los valores Q de *rQ-learning* converjan (figura 5.10). Llegar a las $4,000$ iteraciones le tomó a nuestro método 34 min. Aprendizaje por refuerzo relacional requirió de un total de

23,000 iteraciones y de un tiempo de 4 hrs. 18 min. para generar su política de control de seguimiento.

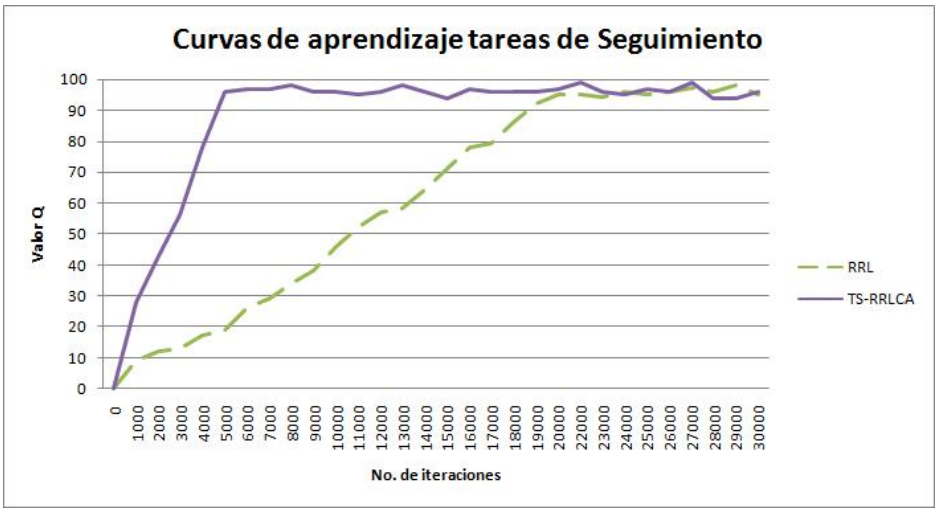


Figura 5.10: Curvas de aprendizaje para las políticas de seguimiento.

El modelo clásico de aprendizaje por refuerzo (RL) y aprendizaje por refuerzo relacional (RRL) presentan las siguientes desventajas con respecto al método propuesto ($TS\text{-}RRLCA$):

- El número de estados es mucho más grande.
- Los tiempos de generación de las políticas son mucho mayores que los tiempos de $TS\text{-}RRLCA$ esto debido principalmente a que clonación de comportamiento reduce significativamente el número de posibles combinaciones de r-estados y r-acciones..
- Las políticas de RL no permiten variaciones en las posiciones de destino (para las tareas de navegación) ni en las posiciones del robot que está siendo perseguido (para las tareas de seguimiento).

Con base en nuestros experimentos se puede afirmar que el método propuesto es capaz de generar políticas de control en relativamente pocas iteraciones y sólo es necesario generar una política de control para la ejecución de diversas tareas de seguimiento y navegación.

Para convertir las políticas con acciones discretas a través del esquema propuesto de regresión pesada local (descrito en la sección 4.4.1) se realizó el correspondiente proceso de triangulación de las posiciones de los robots de las trazas. Si la política relacional con acciones discretas que se quiere convertir a política relacional con acciones continuas es la política de navegación, se utilizan los registros de pares r-estado-r-acción de la base de datos DB_{nav} para este proceso de triangulación. Si se trata de la política de seguimiento se utilizan los registros de la base de datos DB_{seg}. Con estos registros se calcularon las distancias entre el robot en el r-estado actual y los robots de cada una de los registros de las trazas que compartían los mismos valores de r-estado. A cada valor de distancia se le asignó un peso utilizando el *kernel* Gaussiano como función de estimación de pesos. Los pesos se multiplicaron por el valor de las acciones que realizaron los robots de las trazas. La suma de los pesos multiplicados por los valores de las acciones de las trazas genera el valor de la acción continua a realizar en cada estado.

Después de haber generado las políticas con acciones discretas y con acciones continuas se realizaron pruebas con el robot simulado en los ambientes de las figuras 5.2, 5.11, 5.12 y 5.13. El mapa 3 de la figura 5.12 corresponde a la planta baja del edificio Chavira perteneciente a las instalaciones del INAOE. El mapa 4 de la figura 5.13 corresponde al mapa actual del ambiente real en el que se encuentra Kalmancito, el robot real de pruebas. En este ambiente se hicieron las pruebas con el robot real.

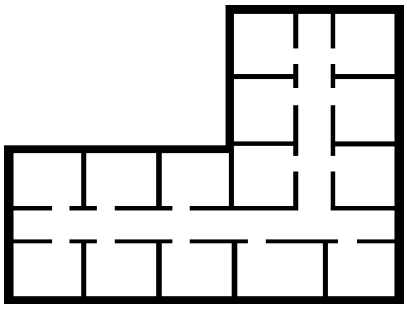


Figura 5.11: Mapa simulado, tamaño 20,0$m.$ × 14,0$m.$ (mapa 2)

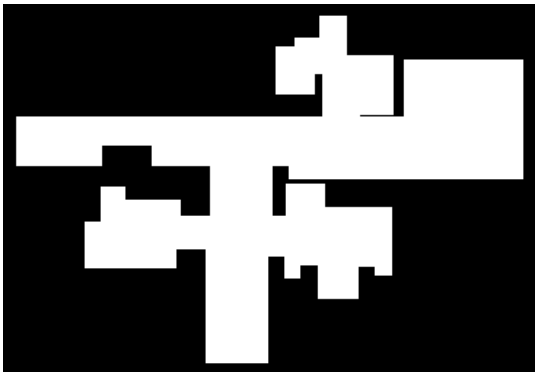


Figura 5.12: Mapa del edificio Chavira del INAOE, tamaño 17,5$m.$ × 12,0$m.$ (mapa 3)

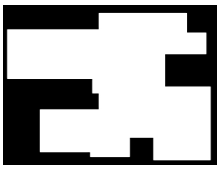


Figura 5.13: Mapa del laboratorio de robótica, tamaño 5,0$m.$ × 5,0$m.$ (mapa 4)

5.3 Pruebas de las políticas

Para evaluar las políticas generadas con *TS-RRLCA* se realizó un conjunto de 40 experimentos, 10 en el mapa 1, 10 en el mapa 2 y otros tantos iguales en los mapas 3 y 4 respectivamente. Cada experimento consta de una tarea de navegación y de una tarea de seguimiento. Y cada tarea fue ejecutada dos veces, la primera utilizando la política con acciones discretas y la segunda con la política que permite ejecutar acciones continuas. Cada tarea tiene una distancia diferente por cubrir y un nivel de dificultad distinto. Las tareas, la distancia recorrida y sus niveles de complejidad se muestran en la tabla 5.1. La descripción de cada nivel de complejidad se muestra en la tabla 5.2.

Cuando no se especifica algún nivel de dificultad, significa que la tarea se llevó a cabo en una misma habitación o pasillo y no se detectó cambio alguno en la ubicación local del robot.

Tabla 5.1: Características de los experimentos realizados.

No. tarea	Mapa 1		Mapa 2		Mapa 3		Mapa 4	
	Dist.	Comp.	Dist.	Comp.	Dist.	Comp.	Dist.	Comp.
1	2 m.		2 m.		2 m.		1 m.	
2	4 m.		4 m.		4 m.		2 m.	
3	6 m.		6 m.	A, B	6 m.	A, B, C	3 m.	A
4	8 m.		8 m.	A, C	8 m.	A, B, C	4 m.	A, C
5	8 m.	A, B	10 m.	A, C	10 m.	A, C	5 m.	A, C
6	9 m.	A, B	12 m.	A, B, C	12 m.	A, B, C	6 m.	A, C
7	9 m.	A, B	14 m.	A, B, C	14 m.	A, B, C	7 m.	A, C
8	10 m.	A, B	16 m.	A, B, C	16 m.	A, B, C	8 m.	A, C
9	10 m.	A, B	18 m.	A, B, C	18 m.	A, B, C	9 m.	A, C
10	12 m.	A, B	20 m.	A, B, C	20 m.	A, C	10 m.	A, C

Tabla 5.2: Descripciones de los niveles de complejidad

Complejidad	**Descripción**
A	El robot se desplaza a lo largo de pasillos
B	El robot atraviesa puertas
C	El robot gira en intersecciones o esquinas

El valor de distancia corresponde a la distancia de *Manhattan*[3] (ec. 5.1) que existe entre el punto en el cual inició la tarea (x_i, y_i) y el punto en el cual terminó (x_f, y_f).

[3]La distancia de *Manhattan* calcula la distancia que se debe recorrer de un punto i a otro punto j si se sigue un camino en forma de celdas o cuadrícula.

$$d(i,j) = |x_i - x_f| + |y_i - y_f| \tag{5.1}$$

Las políticas generadas en el mapa de la figura 5.2 fueron capaces de hacer que el robot desempeñara las tareas de navegación y de seguimiento en los mapas de las figuras 5.11, 5.12 y 5.13. Al realizar los experimentos en estos ambientes desconocidos se presentaron ocasiones en las cuales el robot llegó a un estado desconocido y por lo tanto no supo qué acción ejecutar. Esto ocurre cuando el robot llega a un estado que no había "visto" anteriormente durante la etapa de entrenamiento lo cual es muy comprensible debido a que los ambientes de entrenamiento y pruebas, a pesar de ser del mismo tipo, presentan variaciones importantes en su arquitectura. Uno de los objetivos del presente trabajo es que el robot no tenga que volver a generar una nueva política de navegación cuando llega a un nuevo ambiente, como sería el caso de aprendizaje por refuerzo clásico, sino que re-utilice el conocimiento que previamente (en otro ambiente) había aprendido.

Cuando se presentan estos casos, se tienen dos soluciones posibles. La primera es hacer que el robot realice acciones aleatorias hasta llegar a un estado conocido y de ahi continuar con su política. La segunda es pedir ayuda al usuario el cual, cuando el robot genera una alerta o señal de estado desconocido, proporciona la acción que el robot debe realizar para salir de ese estado. En el presente trabajo se optó por esta segunda opción cada que el robot llegaba a un estado no conocido. Una vez que el usuario indicaba la correspondiente acción para el estado desconocido, los correspondientes pares r-estado-r-acción se almacenaban en su correspondiente DB (DB_{nav} para las trareas de navegación y DB_{seg} para las tareas de seguimiento).

Al final, el total de pares r-estado-r-acción fue de $8,075$ en tareas de navegación y de $10,722$ en tareas de seguimiento, esto tomando en cuenta los valores numéricos. Sin tomar en cuenta estos valores numéricos, es decir, sin descripciones nominales repetidas el total fue de 982 pares para las tareas de navegación y de $1,614$ para las tareas de seguimiento. Esa

situación ocurrió, normalmente, al inicio de los experimentos. Conforme fueron avanzando las pruebas, estos casos no observados fueron disminuyendo gradualmente. La tabla 5.3 muestra el total de ocasiones en las que ocurrieron tales situaciones por mapa, con cada una de las políticas.

Tabla 5.3: Tabla del número de ocasiones por mapa en las cuales el robot solicitó ayuda al usuario.

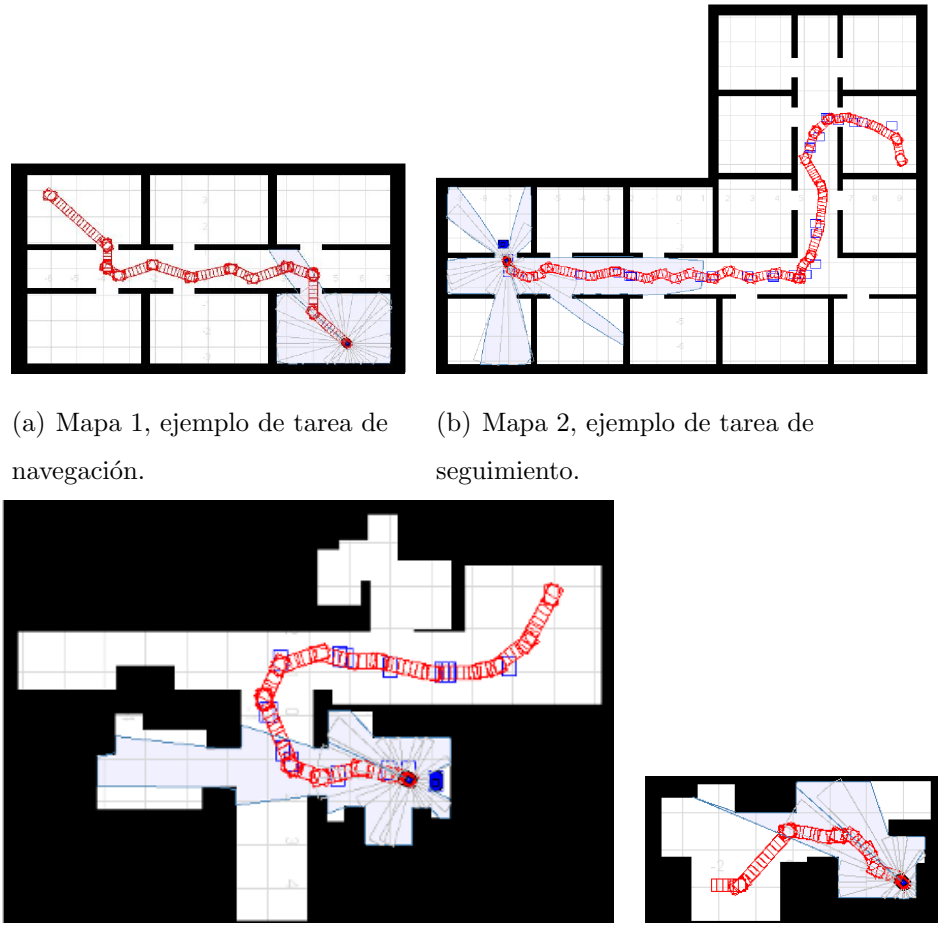
Política	Mapa 1	Mapa 2	Mapa 3	Mapa 4	Total
Navegación	2	6	14	3	25
Seguimiento	7	18	37	9	71

En la figura 5.14 se muestran algunos resultados obtenidos luego de que el robot ejecutara diversas tareas de navegación y de seguimiento haciendo uso de las políticas con acciones discretas. En la figura 5.15 se muestran los resultados de esas mismas tareas pero haciendo uso de políticas con acciones continuas.

En la siguiente sección se presentan evaluaciones de la calidad de las tareas ejecutadas con estas políticas con ambos robots.

5.4 Desempeño de las políticas

Las tareas ejecutadas utilizando las políticas generadas por *TS-RRLCA* se evaluan de tres formas. En la primera se registran los tiempos que toma la ejecución de las tareas haciendo uso de las políticas con acciones discretas contra las que hacen uso de acciones continuas. En la segunda se compara la calidad de las tareas ejecutadas con las políticas de *TS-RRLCA* contra la calidad de las tareas cuando son ejecutadas por el usuario, es decir, se compara la trayectoria seguida por el robot al ejecutar sus tareas siguiendo las políticas generadas con *TS-RRLCA*, contra la trayectoria de traza. Y en la tercera se evalua



(a) Mapa 1, ejemplo de tarea de navegación.

(b) Mapa 2, ejemplo de tarea de seguimiento.

(c) Mapa 3, ejemplo de tarea de seguimiento.

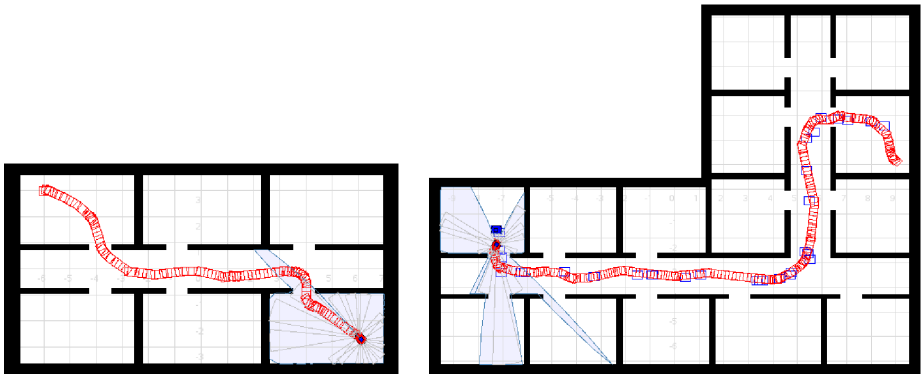
(d) Mapa 4, ejemplo de tarea de navegación.

Figura 5.14: Ejemplos de tareas ejecutadas por el robot, haciendo uso de las políticas con acciones discretas.

el aprovechamiento de los espacios libres del ambiente mientras el robot ejecuta sus tareas.

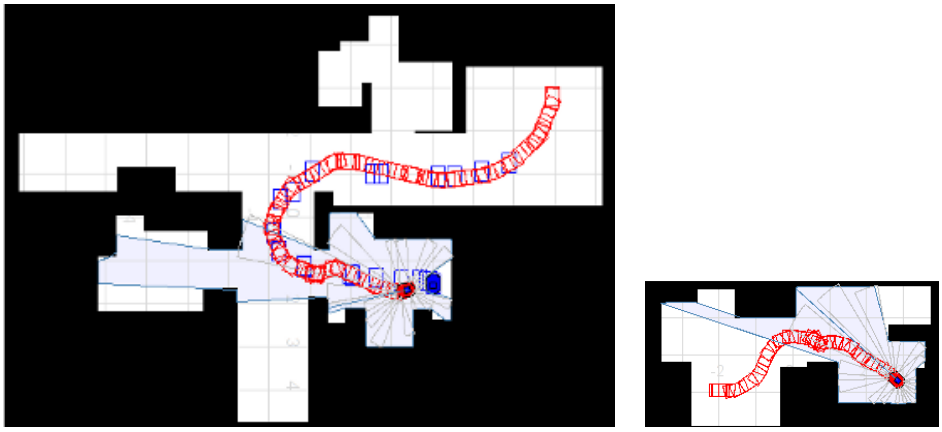
5.4.1 Evaluación tiempo de ejecución de tareas

Con la finalidad de evaluar la diferencia de tiempo entre las tareas ejecutadas con acciones discretas contra las ejecutadas con acciones continuas utilizando las políticas generadas con



(a) Mapa 1, ejemplo de tarea de navegación.

(b) Mapa 2, ejemplo de tarea de seguimiento.



(c) Mapa 3, ejemplo de tarea de seguimiento.

(d) Mapa 4, ejemplo de tarea de navegación.

Figura 5.15: Ejemplos de tareas ejecutadas por el robot, haciendo uso de las políticas con acciones continuas.

TS-RRLCA, se tomó el tiempo que duró cada experimento con el robot real[4] en el ambiente del mapa 3. Como se vió anteriormente, cada tarea de navegación y de seguimiento fue realizada con la política de acciones discretas y posteriormente con la política de acciones

[4]Se presentan los valores de tiempo del robot real para mostrar que la generación de las políticas con acciones continuas, aún con el proceso de regresión pesada local, son más rápidas que las políticas con acciones discretas.

continuas. Se tomó el tiempo de los experimentos de cada tarea de navegación y seguimiento, con cada tipo de política. Dando un total de 40 lecturas de tiempo y de su correspondiente distancia recorrida. La figura 5.16 muestra los valores correspondientes de los tiempos de ejecución de estas políticas[5].

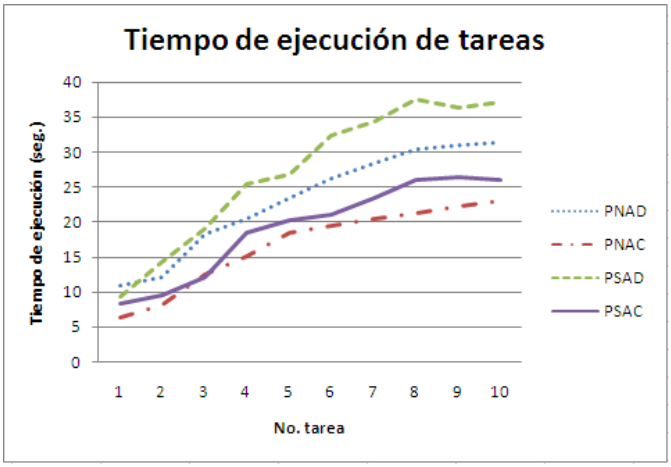


Figura 5.16: Tiempo de ejecución de las políticas de navegación y seguimiento con acciones discretas y continuas haciendo uso del robot real.

Como se puede apreciar en la figura 5.16, las tareas ejecutadas con acciones discretas toman una mayor cantidad de tiempo en ser completadas de forma exitosa que aquellas que son ejecutadas con acciones continuas. Sin embargo, no basta sólo con obtener los tiempos de ejecución de las tareas sino que también se debe obtener una medida o evaluación que indique qué tan bien se han realizado las diferentes tareas.

[5]**PNAD**: Política de Navegación con Acciones Discretas

PNAC: Política de Navegación con Acciones Continuas

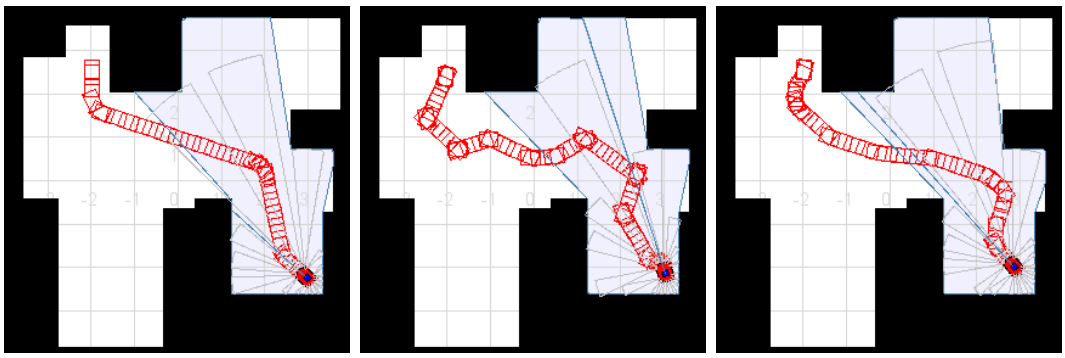
PNAD: Política de Seguimiento con Acciones Discretas

PNAC: Política de Seguimiento con Acciones Continuas

5.4.2 Evaluación de calidad

Las trazas que proporciona el usuario se suponen (casi)perfectas ya que son ejecutadas por personas que conocen bien cómo realizar las tareas. En el presente trabajo se utiliza información de estas trazas y una parte importante es analizar qué tan bien se imita este comportamiento, i.e., qué tan bien se aprovecha el conocimiento del usuario.

A continuación se muestra un ejemplo de una de las tareas de navegación realizadas por el usuario (figura 5.17(a)), de la misma tarea de navegación ejecutada con acciones discretas (figura 5.17(b)) y de la misma tarea ejecutada con acciones continuas (figura 5.17(c)).



(a) Tarea de navegación ejecutada por el usuario.

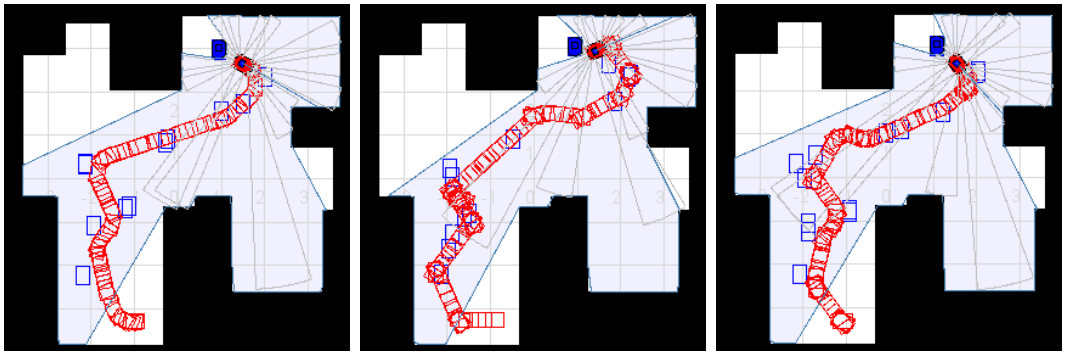
(b) Tarea de navegación ejecutada con acciones discretas.

(c) Tarea de navegación ejecutada con acciones continuas.

Figura 5.17: Resultados de las políticas de navegación respecto a las tareas ejecutadas por el usuario.

De igual forma se muestra un ejemplo de una de las tareas de seguimiento realizadas por el usuario (figura 5.18(a)), de esa misma tarea de seguimiento ejecutada con acciones discretas (figura 5.18(b)) y de la misma tarea ejecutada con acciones continuas (figura 5.18(c)).

La diferencia o medida de calidad entre las tareas realizadas por el usuario y las de las políticas, se estima por diferencias de error cuadrático. Se utiliza esta medida debido a que es una de las más sencillas de implementar y de las que presenta mejores resultados



(a) Tarea de seguimiento ejecutada por el usuario.

(b) Tarea de seguimiento ejecutada con acciones discretas.

(c) Tarea de seguimiento ejecutada con acciones continuas.

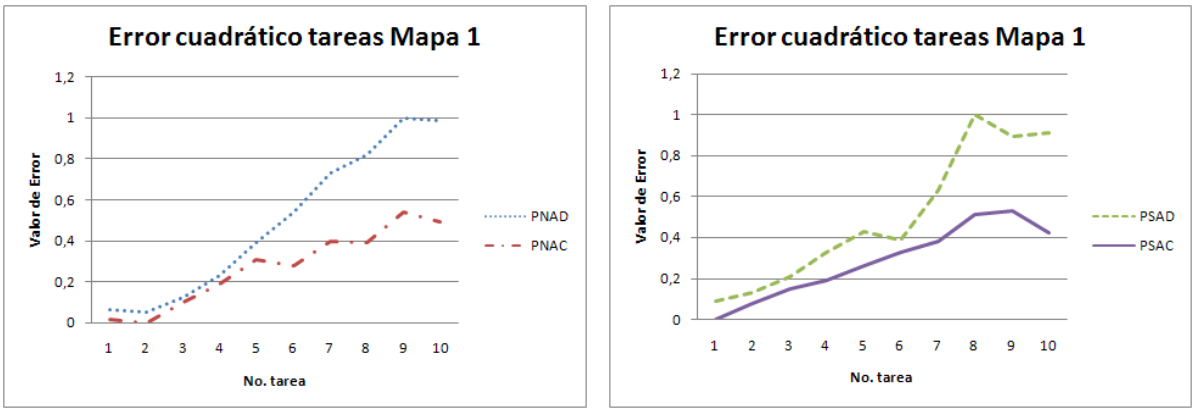
Figura 5.18: Resultados de las políticas de seguimiento respecto a las tareas ejecutadas por el usuario.

ya que permite establecer la diferencia absoluta que existe entre un punto o posición con respecto de otra. Es por ello que con esta medida se estima la diferencia que existe entre las posiciones del robot registradas durante las trazas contra las posiciones del robot registradas cuando se ejecutan por las políticas. Debido a que en la etapa de generación de trazas, sólo se generaron veinte de estas, fue necesario que el usuario también realizara los 120 experimentos reportados anteriormente. Esto se hizo con la finalidad de obtener las posiciones absolutas del robot en las trazas y poder ser comparadas con las de las políticas. El error cuadrático E entre estas posiciones se obtiene con la ecuación 5.2:

$$E = \frac{1}{2}\sum_{x \in D}(f(x) - \hat{f}(x))^2 \tag{5.2}$$

Donde x representa el punto sobre el cual se quiere obtener el error al cuadrado, $f(x)$ representa el valor de $f(x)$ observado en la traza y $\hat{f}(x)$ representa el valor de $f(x)$ observado con las políticas. Posteriormente estos valores fueron graficados. Las gráficas del error cuadrado, de las tareas de navegación y seguimiento por mapa, se muestran en las figuras

5.19, 5.20, 5.21 y 5.22. Seguido de cada figura se encuentra su correspondiente tabla[6] con los valores normalizados de los errores cuadráticos obtenidos de comparar la calidad de las trazas contra la calidad de cada política. En cada tabla también se agrega una fila con la suma de los valores de error totales.



(a) Error cuadrático, tareas de navegación. (b) Error cuadrático, tareas de seguimiento.

Figura 5.19: Gráficas de error cuadrático de las tareas ejecutadas en el mapa 1.

Como se puede apreciar, el error cuadrático de las políticas con acciones continuas, respecto a las trazas, es menor que el error cuadrático de las políticas con acciones discretas. La diferencia en el error cuadrático de las tareas del robot real ejecutadas con las políticas con acciones continuas tiende a ser mucho mayor comparándola contra el error cuadrático de las políticas con acciones discretas. Esto se debe principalmente a los errores de odometría que se acarrean entre la ejecución de una acción discreta y otra.

[6]**PNAD**: Política de Navegación con Acciones Discretas

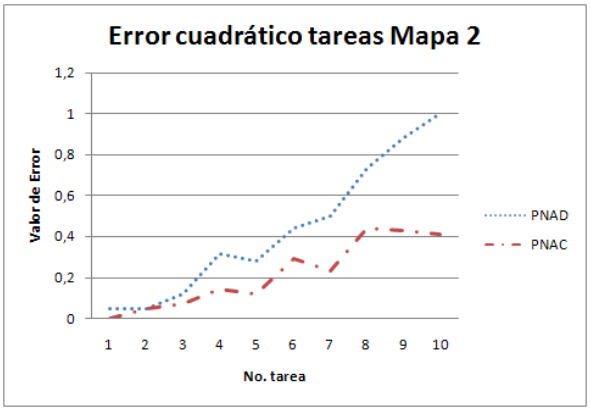
PNAC: Política de Navegación con Acciones Continuas

PNAD: Política de Seguimiento con Acciones Discretas

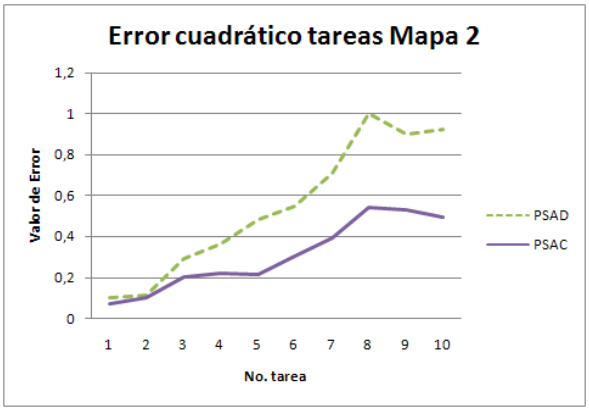
PNAC: Política de Seguimiento con Acciones Continuas

Tabla 5.4: Tabla de errores cuadráticos entre las tareas ejecutadas por el usuario y las tareas de las políticas en el mapa 1.

No. mapa	No. tarea	Error PNAD	Error PNAC	Error PSAD	Error PSAC
Mapa 1	1	0.06	0.02	0.09	0.00
	2	0.05	0.00	0.13	0.08
	3	0.12	0.10	0.21	0.15
	4	0.23	0.19	0.33	0.19
	5	0.39	0.31	0.43	0.26
	6	0.54	0.28	0.37	0.33
	7	0.73	0.40	0.63	0.38
	8	0.82	0.39	1.00	0.51
	9	1.00	0.54	0.89	0.53
	10	0.92	0.49	0.88	0.42
	Total	**4.86**	**2.72**	**4.96**	**2.85**



(a) Error cuadrático, tareas de navegación.

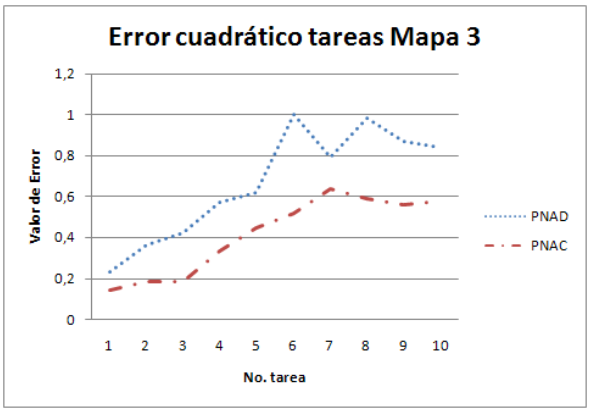


(b) Error cuadrático, tareas de seguimiento.

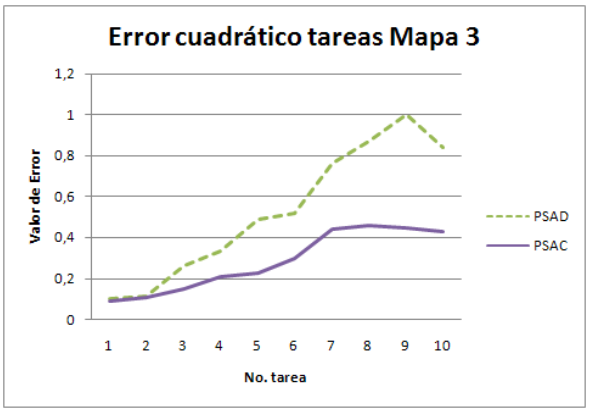
Figura 5.20: Gráficas de error cuadrático de las tareas ejecutadas en el mapa 2.

Tabla 5.5: Tabla de errores cuadráticos entre las tareas ejecutadas por el usuario y las tareas de las políticas en el mapa 2.

No. mapa	No. tarea	Error PNAD	Error PNAC	Error PSAD	Error PSAC
Mapa 2	1	0.05	0.00	0.10	0.07
	2	0.05	0.05	0.11	0.10
	3	0.12	0.07	0.29	0.20
	4	0.32	0.14	0.36	0.22
	5	0.28	0.12	0.48	0.21
	6	0.44	0.29	0.55	0.30
	7	0.50	0.23	0.71	0.39
	8	0.73	0.44	1.00	0.54
	9	0.88	0.43	0.90	0.53
	10	1.00	0.56	0.92	0.49
	Total	**4.37**	**2.33**	**5.42**	**3.05**



(a) Error cuadrático, tareas de navegación.

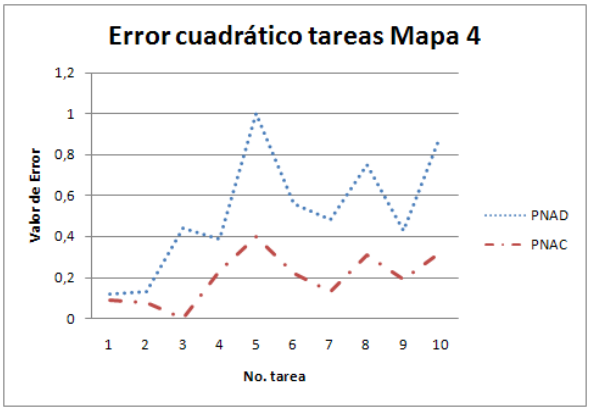


(b) Error cuadrático, tareas de seguimiento.

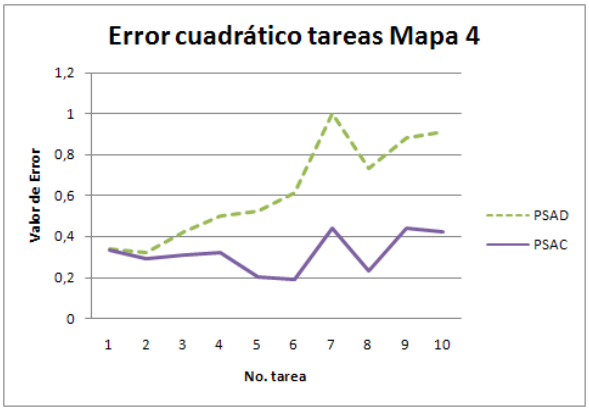
Figura 5.21: Gráficas de error cuadrático de las tareas ejecutadas en el mapa 3.

Tabla 5.6: Tabla de errores cuadráticos entre las tareas ejecutadas por el usuario y las tareas de las políticas en el mapa 3.

No. mapa	No. tarea	Error PNAD	Error PNAC	Error PSAD	Error PSAC
Mapa 3	1	0.16	0.16	0.10	0.09
	2	0.21	0.18	0.11	0.12
	3	0.17	0.13	0.26	0.18
	4	0.32	0.36	0.33	0.25
	5	0.47	0.6	0.49	0.29
	6	0.64	0.52	0.52	0.32
	7	1.00	0.54	0.76	0.59
	8	0.93	0.53	1.00	0.73
	9	0.82	0.66	0.87	0.65
	10	0.79	0.41	0.91	0.69
	Total	**5.53**	**4.09**	**5.35**	**3.91**



(a) Error cuadrático, tareas de navegación.



(b) Error cuadrático, tareas de seguimiento.

Figura 5.22: Gráficas de error cuadrático de las tareas ejecutadas en el mapa 4.

Tabla 5.7: Tabla de errores cuadráticos entre las tareas ejecutadas por el usuario y las tareas de las políticas en el mapa 4.

No. mapa	No. tarea	Error PNAD	Error PNAC	Error PSAD	Error PSAC
Mapa 4	1	0.12	0.09	0.34	0.33
	2	0.13	0.08	0.32	0.29
	3	0.44	0.00	0.42	0.31
	4	0.42	0.23	0.50	0.32
	5	1.00	0.40	0.22	0.20
	6	0.69	0.22	0.61	0.19
	7	0.63	0.23	1.00	0.44
	8	0.75	0.31	0.73	0.23
	9	0.73	0.29	0.88	0.44
	10	0.89	0.32	0.91	0.55
	Total	**5.8**	**2.17**	**5.93**	**3.3**

5.4.3 Evaluación de espacios libres utilizados

En esta evaluación se supone que las tareas que realizan los robots de servicio en los ambientes de tipo hogar u oficina como son, por ejemplo, llevar objetos de un lugar a otro (libros, revistas, documentos, medicamentos, alimentos, etc.), seguir personas o entregar mensajes, deben de ser ejecutadas utilizando los espacios libres del ambiente. Es decir deben ejecutar sus labores de servicio procurando desplazarse a través de los espacios por los cuales hay menores posibilidades de chocar contra algún objeto del ambiente como puede ser alguna pared, alguna mesa, alguna silla o inclusive contra alguna persona, que impida a estos robots de servicio completar sus tareas.

Para evaluar el espacio libre utilizado, se utilizó el método reportado en (Romero et al.

2001) en donde se enseña a navegar a un robot en un mapa de celdas de un ambiente determinado pero, en este esquema, a cada celda se le asigna un valor de ocupación dependiendo de qué tan cerca o lejos se encuentra de algún elemento u obstáculo. Dependiendo del valor de ocupación se generan diversos tipos de regiones en el mapa como son región ocupada, región en zona de peligro, región en zona de prevención y región desocupada. Una vez generadas las regiones de ocupación, se enseña al robot (utilizando MDP's) a desplazarse a través de las celdas desocupadas. Estas celdas son las que suponen las mejores trayectorias de desplazamiento. Con esto el robot tiende a desplazarse a través de aquellos espacios que representan menor peligro o conflicto en la ejecución de sus tareas.

En la figura 5.23 se muestran los mapas de los ambientes de pruebas presentados anteriormente, con sus respectivas celdas de ocupación. Cada tono de color del mapa representa la cercanía de la celda con respecto a algún objeto o elemento del ambiente. Si la celda es de color negro la celda está ocupada, si es de color gris la celda se encuentra cercana a una celda ocupada (zona de peligro), si es de color gris claro es una celda de prevención y si es de color blanco es una celda desocupadas.

En esta evaluación se asigna un valor de castigo o peso a cada tipo de celda, a las celdas desocupadas no se les asigna peso alguno, a las celdas en zona de prevención se les asigna un peso de -1, a las celdas en zona de peligro se les asigna un peso de -3 y, cuando el robot llega a una celda ocupada se le da un valor de -100. Cada celda es de $0{,}5m. \times 0{,}5m.$ Cada vez que el robot se encuentra en una celda los valores de los pesos se suman.

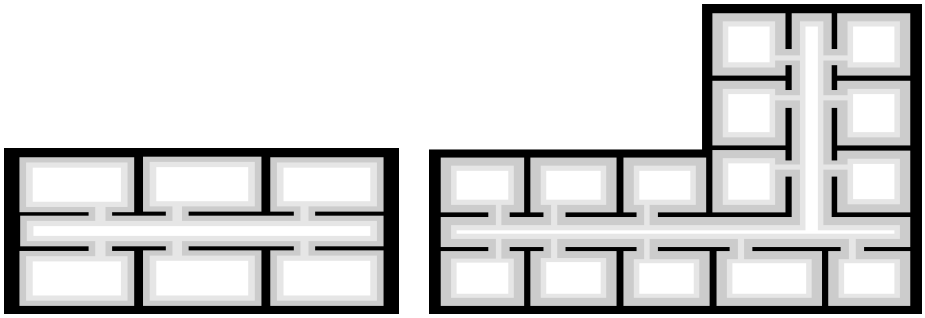
Las gráficas de pesos o valores de castigo asignados a las tareas de navegación y seguimiento se muestran en la figuras 5.24, 5.25, 5.26 y 5.27. Seguido de cada gráfica se muestra una tabla[7] con los valores de los resultados de la suma de los pesos obtenidos en ambas

[7]**PNAD**: Política de Navegación con Acciones Discretas

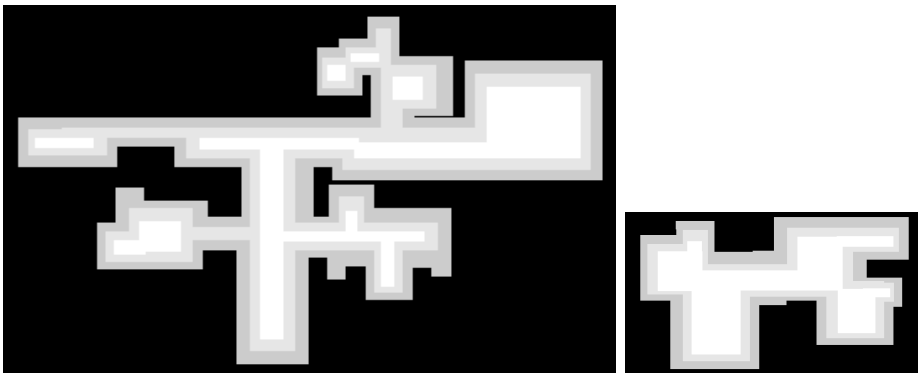
PNAC: Política de Navegación con Acciones Continuas

PNAD: Política de Seguimiento con Acciones Discretas

PNAC: Política de Seguimiento con Acciones Continuas



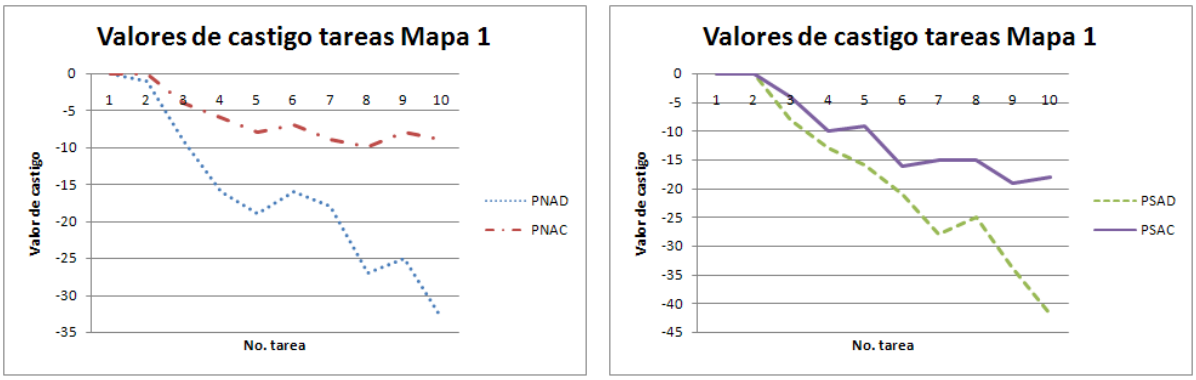
(a) Mapa 1. (b) Mapa 2.



(c) Mapa 3. (d) Mapa 4.

Figura 5.23: Mapas de ocupación.

tareas y con cada tipo de política. Además, en cada tabla se muestra una fila con los valores de castigo totales.



(a) Valores de castigo, tareas de navegación.

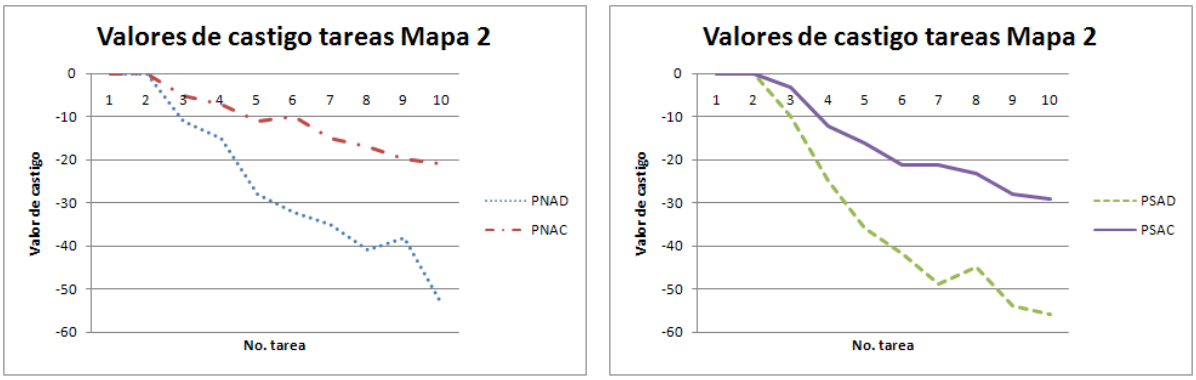
(b) Valores de castigo, tareas de seguimiento.

Figura 5.24: Gráficas de valores de castigo de las tareas ejecutadas en el mapa 1.

Tabla 5.8: Tabla de pesos o valores de castigo asignados a la ejecución de las tareas del mapa 1.

No. Mapa	No. tarea	Error PNAD	Error PNAC	Error PSAD	Error PSAC
Mapa 1	1	0	0	0	0
	2	-1	0	0	0
	3	-9	-4	-8	-4
	4	-16	-6	-23	-12
	5	-19	-10	-26	-11
	6	-16	-9	-31	-18
	7	-18	-12	-38	-17
	8	-27	-12	-35	-17
	9	-25	-8	-49	-21
	10	-33	-16	-57	-26
	Total	**-164**	**-77**	**-267**	**-126**

Como se puede apreciar, las políticas con acciones discretas reciben mayores valores de



(a) Valores de castigo, tareas de navegación.

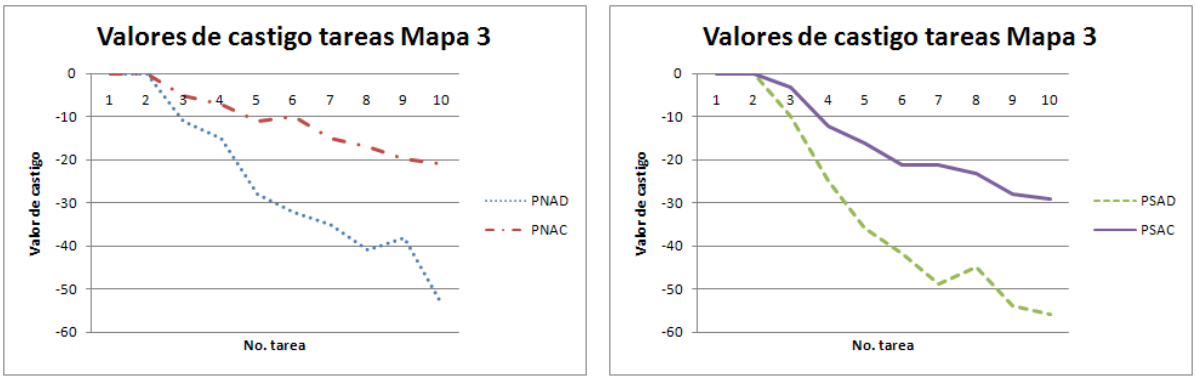
(b) Valores de castigo, tareas de seguimiento.

Figura 5.25: Gráficas de valores de castigo de las tareas ejecutadas en el mapa 2.

Tabla 5.9: Tabla de pesos o valores de castigo asignados a la ejecución de las tareas del mapa 2.

No. Mapa	No. tarea	Error PNAD	Error PNAC	Error PSAD	Error PSAC
Mapa 2	1	0	0	0	0
	2	0	0	0	0
	3	-11	-5	-10	-3
	4	-15	-7	-25	-14
	5	-28	-11	-36	-18
	6	-32	-10	-42	-22
	7	-35	-15	-49	-21
	8	-41	-18	-45	-23
	9	-38	-21	-54	-29
	10	-53	-24	-43	-37
	Total	**-253**	**-111**	**-304**	**-167**

castigo. Esto se debe a que las trayectorias que siguen, tienden a aproximarse a los espacios o zonas más concurridas del ambiente en lugar de aprovechar las zonas despejadas donde es



(a) Valores de castigo, tareas de navegación.

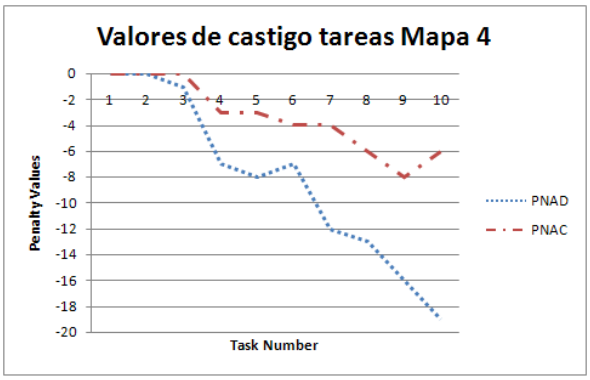
(b) Valores de castigo, tareas de seguimiento.

Figura 5.26: Gráficas de valores de castigo de las tareas ejecutadas en el mapa 3.

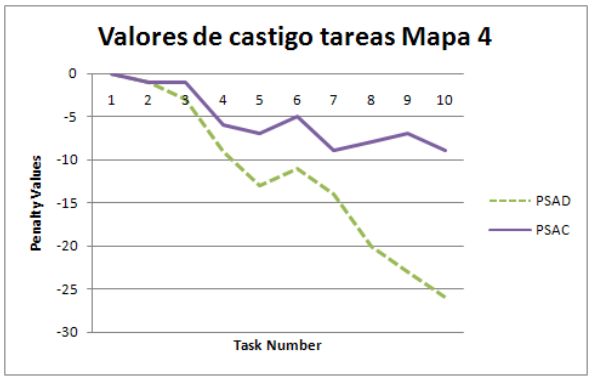
Tabla 5.10: Tabla de pesos o valores de castigo asignados a la ejecución de las tareas del mapa 3.

No. Mapa	No. tarea	Error PNAD	Error PNAC	Error PSAD	Error PSAC
Mapa 3	1	-4	-3	-3	-4
	2	-5	-3	-5	-5
	3	-16	-12	-16	-9
	4	-21	-13	-31	-21
	5	-33	-18	-42	-22
	6	-38	-16	-47	-27
	7	-41	-19	-55	-28
	8	-49	-23	-50	-25
	9	-43	-28	-62	-28
	10	-58	-33	-49	-44
	Total	**-304**	**-165**	**-357**	**-209**

menos probable la colisión con algún objeto o elemento del ambiente que impida al robot completar su tarea. Las políticas con acciones discretas no tienden a incrementar los valores



(a) Valores de castigo, tareas de navegación.



(b) Valores de castigo, tareas de seguimiento.

Figura 5.27: Gráficas de valores de castigo de las tareas ejecutadas en el mapa 4.

Tabla 5.11: Tabla de pesos o valores de castigo asignados a la ejecución de las tareas del mapa 4.

No. Mapa	No. tarea	Error PNAD	Error PNAC	Error PSAD	Error PSAC
Mapa 4	1	0	0	0	0
	2	0	0	-1	-1
	3	-1	0	-3	-1
	4	-7	-3	-9	-6
	5	-8	-3	-13	-7
	6	-7	-4	-11	-5
	7	-12	-4	-14	-9
	8	-13	-6	-20	-8
	9	-16	-8	-23	-7
	10	-19	-6	-26	-9
	Total	**-83**	**-34**	**-120**	**-53**

de castigo inclusive con los experimentos realizados con el robot real.

En este capítulo se presentaron los experimentos realizados por un robot móvil el cual

utiliza las políticas de control de navegación y de seguimiento generadas con el método propuesto. Se comparó al método propuesto con aprendizaje por refuerzo clásico y con aprendizaje por refuerzo relacional en cuanto al número de iteraciones que toma a cada método generar sus políticas. Al método propuesto le toma significativamente menos tiempo generar sus políticas de control que a los métodos de aprendizaje por refuerzo clásico y relacional, además de que las políticas generadas con el presente método pueden ser re-utilizadas o transferibles y las de aprendizaje por refuerzo clásico no. Se mostró cómo las políticas generadas con *TS-RRLCA* pueden ser transferidas a diferentes ambientes tipo hogar u oficina con paredes planas, puertas, pasillos y esquinas.

Las políticas con acciones continuas mostraron mejorar los tiempos de ejecución de las tareas al ser comparadas contra las políticas con acciones dicretas. Además se comparó el desempeño de las tareas ejecutadas con las políticas con acciones continuas generadas contra esas mismas tareas cuando son ejecutadas por humanos. En general se observó que las políticas generadas con el método propuesto son capaces de "imitar" de forma aceptable el comportamiento de los humanos cuando realizan sus tareas.

Finalmente, las políticas con acciones continuas muestran que el robot tiende a utilizar una mayor parte del espacio libre del ambiente comparado contra las políticas con acciones discretas; dando como resultado tareas con mayor posibilidad de éxito al evitar que el robot se desplace a lo largo de espacios muy concurridos o saturados del ambiente donde las posibilidades de alguna colisión son mucho mayores.

6

Conclusiones y Perspectivas

6.1 Síntesis

En los últimos años se ha observado un incremento o tendencia en el uso de robots de servicio. Sin embargo, su completa aceptación e incorporación requiere de métodos de adaptación y aprendizaje de nuevas tareas. Para ello se han utilizado distintas técnicas y algoritmos de inteligencia artificial y aprendizaje computacional y entre las más utilizadas se encuentra el aprendizaje por refuerzo. Desafortunadamente, los espacios de estados en robótica son muy grandes, llegando a tener miles o millones de combinaciones posibles de pares estado-acción. Además, se requiere de la ejecución de acciones continuas para mejorar la calidad en la ejecución de las tareas del robot.

Por lo general, el uso de los métodos tradicionales de aprendizaje por refuerzo se han visto limitados debido a que no permite generar políticas en espacios de búsqueda continuos o muy grandes, y a que sólo permite ejecutar acciones discretas. Para solucionar esto se han propuesto diversas estrategias como el aprendizaje por refuerzo con representaciones relacionales. Sin embargo, la mayoría de los métodos requieren de mucho tiempo de entrenamiento y muchos ejemplos para generar estas representaciones y por lo general se hace

uso de acciones discretas. Los enfoques que generan acciones continuas, por lo general, son computacionalmente caros y las soluciones no son transferibles a otros ambientes similares.

En esta tesis se utilizan representaciones relacionales de términos de alto nivel como habitaciones, puertas, paredes, pasillos, etc., permitiendo un fácil entendimiento de conceptos y la transferencia de conocimiento. Desafortunadamente, aunque el conocimiento pueda ser transferible o fácil de entender/interpretar, los métodos de aprendizaje por refuerzo son lentos y hacen uso de acciones discretas. Para abordar esta problemática en este trabajo de tesis se siguieron dos estrategias. La primera fue utilizar clonación de comportamiento para proporcionar una guía al método de aprendizaje por refuerzo y así reducir el espacio de búsqueda. La segunda fue dividir el método en dos fases, una fase que genera de forma rápida una política de control con acciones discretas, basada en una discretización sobre relaciones, y otra de regresión pesada local que de forma rápida genera acciones continuas.

Se demostró que el método es capaz de funcionar tanto en simulación (*player-stage*) como en un robot de servicio real (un *GuiaBot* de *ActivMedia*). Con ambos robots las políticas fueron capaces de lograr sus objetivos, i.e., llegar a una meta. También se demostró que las acciones continuas, en comparación con acciones discretas, generan trayectorias que recorren menores distancias, son ejecutadas en menos tiempo, acumulan menos errores y son más seguras para la integridad del robot mismo y de los diversos elementos del ambiente.

6.2 Aportaciones

La aportación principal del presente trabajo es un método para generar políticas de control relacionales con acciones continuas, que se generan en pocas iteraciones y que pueden ser trasladadas a diversos ambientes con variaciones en las posiciones de las metas u objetivos de las tareas.

El método se divide en dos fases, una que utiliza aprendizaje por refuerzo relacional y otra que utiliza una regresión pesada local. Las características a resaltar de este método son:

- Se permite transformar los datos de los sensores del robot en representaciones relacionales abstractas y basadas en puertas, paredes, pasillos, etc.
- Estas representaciones relacionales permiten la fácil inclusión de nuevas relaciones para la caracterización de nuevos elementos/estados.
- Se generan políticas transferibles basadas en puertas, paredes, esquinas, etc.
- Las políticas permiten la ejecución de acciones continuas.
- Las tareas realizadas por estas políticas requieren menor tiempo de ejecución que las políticas discretas.
- Las políticas continuas generan trayectorias más cortas que las políticas discretas.
- Las políticas continuas generan trayectorias con menores errores que las discretas al ser comparadas contra tareas ejecutadas por humanos.
- Las tareas basadas en las políticas continuas tienden a realizarse a través de espacios más seguros o despejados del ambiente que las políticas discretas.
- Las políticas pueden ser utilizadas por robots tanto simulados como reales.

6.3 Conclusiones y perspectivas

En el presente trabajo de tesis se desarrolló un método en dos fases llamado $TS\text{-}RRLCA$ para la generación de políticas de control relacionales con acciones continuas. Este método aprovecha las ventajas, como son la simplicidad y autonomía, de los algoritmos de aprendizaje por refuerzo (principalmente *Q-learning*) y ataca sus desventajas como son los tiempos largos de generación de las políticas de control, la imposibilidad de re-utilizar políticas en ambientes diferentes pero similares y la ejecución de acciones discretas.

Con este método se demostró que es posible enseñar a un robot móvil cómo realizar tareas de navegación y de seguimiento a partir de un conjunto de trazas proporcionadas por un usuario.

El método fue probado generando políticas de control de robots móviles de servicio, para la ejecución de tareas de navegación y seguimiento en simulación y con un robot real; un *Guiabot* de *ActivMedia*.

Las tareas ejecutadas con estas políticas fueron capaces de ser re-utilizadas en ambientes diferentes. La calidad en la ejecución de las tareas fue comparable con la calidad de las tareas ejecutadas por el usuario en las trazas. Esta comparación se realizó tomando en cuenta aspectos como distancias recorridas, tiempo y seguridad (evadir las zonas o secciones del ambiente ocupadas por algún elemento u obstáculo, procurando deplazarse por las zonas libres del mapa).

El aprendizaje de estas tareas toma aproximadamente un 40 % del tiempo que toma a aprendizaje por refuerzo relacional generar las mismas iteraciones. Además, las políticas generadas con nuestro método son significativamente mejores.

En futuros desarrollos se empleará el método desarrollado para tareas que requieran mayores grados de precisión como son la toma y manipulación de objetos o para dominios diferentes, como por ejemplo el simulador de avión de alto desempeño presentado en (Morales y Sammut 2004). También se piensa utilizar otro método de regresión como son los procesos Gaussianos que han demostrado llevar a cabo mejores aproximaciones de funciones continuas que la regresión pesada local pero que también requieren una mayor cantidad de recursos (Nguyen-Tuong y Peters 2008). Además de esto, se piensa desarrollar una estrategia de búsqueda o exploración del ambiente para completar nuestro conjunto de pares estado-acción. Esto con la finalidad de cubrir una mayor cantidad de situaciones o estados no vistos en los ejemplos de las trazas. Finalmente, se planea integrar comandos de voz para guiar al robot cada que llegue a estados no conocidos, esto con la finalidad de proporcionar un mejor esquema de control, más sencillo de utilizar y más natural para los usuarios que una consola

de una computadora o que un *joystick*.

Publicaciones derivadas de la tesis

Julio C. Zaragoza and Eduardo F. Morales.: 2009, A Two-Stage Relational Reinforcement Learning with Continuous Actions for Real Service Robots, *Proc. of the 8th Mexican International Conference on Artificial Intelligence 2009,* LNAI 5845, Hernández A. et al. (eds). Springer-Verlag, Berlin.

Bibliografía

Bratko, I., Urbancic, T. y Sammut, C.: 1998, Behavioural cloning of control skill, *Machine Learning and Data Mining* pp. 335–351.

Cocora, A., Kersting, K., Plagemann, C., Burgard, W. y Raedt, L. D.: 2006, Learning relational navigation policies, *Journal of Intelligent and Robotic System* pp. 2792–2797.

Conn, K. y Peters, R. A.: 2007, Reinforcement learning with a supervisor for a mobile robot in a real-world environment, *Proc. of the 2007 IEEE International Symposium on Computational Intelligence in Robotics and Automation Jacksonville, FL, USA* pp. 73–78.

Cuaya, G. y Munoz, A.: 2007, Control de un robot hexápodo basado en procesos de decisión de markov, *Memorias del Primer Encuentro de Estudiantes en Ciencias de la Computación; Centro de Investigación en Computación del Instituto Politécnico Nacional, México* .

Dzeroski, S., Raedt, L. D. y Driessens, K.: 1998, Relational reinforcement learning, *Machine Learning* **43**(1-2), 7–52.

Gaskett, C., Wettergreen, D., Zelinsky, A. y Zelinsky, E.: 1999, Q-learning in continuous state and action spaces, *Proc. of the Australian Joint Conference on Artificial Intelligence* **417-428**.

Govea, B. A. V. y Morales, E. F.: 2007, Learning navigation teleo-operators with behavioural cloning, pp. 209–212.

Hasselt, H. V. y Wiering, M. A.: 2007, Reinforcement learning in continuous action spaces, *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Acropolis Convention Center, Niza, France* pp. 272–279.

Hernández, S. F. y Morales, E. F.: 2006, Global localization of mobile robots for indoor environments using natural landmarks, *Proc. of the IEEE International Conference on Robotics, Automation and Mechatronics (RAM)* pp. 29–30.

Herrera-Vega, J.: 2009, Mobile robot localization in topological maps using visual information, *Tesis de Maestría, BUAP* .

Klingspor, V., Morik, K. y Rieger, A. D.: 1996, Learning concepts from sensor data of a mobile robot, *Machine Learning* **23**(2-3), 305–332.
URL: *citeseer.ist.psu.edu/klingspor96learning.html*

Lee., H., Shen, Y., Yu, C.-H., Singh, G. y Andrew, Y.Ñ.: 2006, Quadruped robot obstacle negotiation via reinforcement learning, *Proc. of the IEEE International Conference on Robotics and Automation, ICRA* pp. 3003–3010.

Lin, L.-J.: 1993, Reinforcement learning for robots using neural networks, *Proc. of the IEEE International Conference on Robotics and Automation, ICRA* .

Mahadevan, S. y Connell, J.: 1991, Automatic programming of behavior-based robots using reinforcement learning, *Proc. of the Ninth National Conference on Artificial Intelligence Anaheim, CA.* pp. 768–773.

Mataric, J. M.: 1994, Reward functions for accelerated learning, *Proc. of the Eleventh International Conference on Machine Learning* pp. 181–189.

Morales, E. F.: 2005, Scaling up reinforcement learning with a relational representation, *Workshop on Adaptability in Multi-agent Systems (AORC-2003)* .

Morales, E. F. y Sammut, C.: 2004, Learning to fly by combining reinforcement learning with behavioural cloning, *Proc. of the Twenty-First International Conference on Machine Learning (ICML-04)* p. 76.

Nguyen-Tuong, D. y Peters, J.: 2008, Local gaussian process regression for real-time model-based robot control, *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Acropolis Convention Center, Nice, France* pp. 22–26.

Peters, J., Vijayakumar, S. y Schaal, S.: 2003, Reinforcement learning for humanoid robotics, *Proc. of the Third IEEE-RAS International Conference on Humanoid Robots, Karlsruhe, Germany* pp. 29–30.

R., V., B., G. y A., H.: 2003, On device abstractions for portable, reusable robot code, *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), Las Vegas, Nevada* pp. 11–15.

Romero, L., Morales, E. F. y Sucar, L. E.: 2001, An exploration and navigation approach for indoor mobile robots considering sensor's perceptual limitations, *Proc. of the IEEE International Conference on Robotics and Automation, ICRA* **3**, 3092–3097.

Russell, S. y Norvig, P.: 2003, *Artificial Intelligence: A Modern Approach.*, Prentice Hall.

Schaal, S. y Atkeson, C. G.: 1991, Robot juggling: An implementation of memory-based learning, *Control Systems Magazine* **14**, 57–71.

Smart, W. D.: 2002, Making reinforcement learning work on real robots, *Department of Computer Science at Brown University, Providence, Rhode Island* p. 147.

Smart, W. D. y Kaelbling, L. P.: 2002, Effective reinforcement learning for mobile robots, *Proc. of the 2002 IEEE International Conference on Robotics and Automation, Washington, DC, USA* **4**, 3404–3410.

Stone, P.: 1997, Layered learning in multiagent systems, *In AAAI/IAAI* p. 819.

Sutton, R. S. y Barto, A. G.: 1998, *Reinforcement Learning: An Introduction.*, MIT Press, Cambridge, MA.

Torrey, L., Shavlik, J., Walker, T. y Maclin, R.: 2008, Relational macros for transfer in reinforcement learning, *Inductive Logic Programming* pp. 254–268.

Walker, T., Torrey, L., Shavlik, J. y Maclin, R.: 2008, Building relational world models for reinforcement learning, *Springer-Verlag Berlin Heidelberg* .

Wang, Y., Huber, M., Papudesi, V.Ñ. y Cook, D. J.: 2003, User-guided reinforcement learning of robot assistive tasks for an intelligent environment, *Proc. of the The 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, USA* **1**, 27–31.

Watkins, C.: 1989, Learning from delayed rewards, *PhD Thesis, University of Cambridge, England*

.