Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Aprendizaje por Refuerzo en Robots Autónomos con percepción basada en visión

Tesis presentada para obtener el título de Licenciado en Ciencias de la Computación de la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires

Abril de 2007

Sol Pedre
Pablo Esteban De Cristóforis

Director: Dr. Juan Miguel Santos

Codirector: Lic. Diego Ariel Bendersky

Jurado: Dra. María Juliana Gambini, Dr. Juan Miguel Santos y Dr. Silvano Zanutto.

Lugar de Trabajo: Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires.

A los estudiantes y docentes que luchan por defender la Educación Pública.

A Carlos Alberto Fuentealba.

Resumen

Un modelo de aprendizaje que ha impactado en el campo de los robots autónomos es el Aprendizaje por Refuerzo. En este trabajo estudiamos la utilización de Aprendizaje por Refuerzo en problemas para los cuales se tiene una percepción basada en visión. El espacio de estados de las imágenes es demasiado grande para poder aplicar técnicas de Aprendizaje por Refuerzo directamente. Surge entonces la necesidad de mapear el espacio de imágenes a una representación cuyo cardinal haga factible el uso de técnicas de Aprendizaje por Refuerzo como Q-Learning.

Para solucionar este problema propusimos un método que utiliza la transformada lineal de Hough para detectar las rectas de las imágenes sensadas. A partir de una representación basada en una cantidad pequeña de rectas inferidas de las imágenes, obtenemos la información necesaria del entorno y reducimos el cardinal de estados lo suficiente, como para poder aplicar el algoritmo de Q-Learning. Además, para tratar el problema de la alta cardinalidad del espacio estado-acción durante la exploración propusimos una técnica de exploración dirigida libre de modelo.

Para verificar el método propuesto realizamos varias experiencias. Primero, desarrollamos un entorno de simulación para poner a prueba el método y ajustar los valores de los parámetros involucrados. Luego, realizamos experiencias con un robot real. Los resultados obtenidos en ambos casos fueron muy satisfactorios. Por último, presentamos las conclusiones y las posibles líneas de investigación que permitirían continuar este trabajo en el futuro.

Agradecimientos

En primer término, quisieramos expresar nuestro más profundo y sincero agradecimiento a nuestros directores de tesis: Juan Santos y Diego Bendersky, por guiarnos durante todo el trabajo con mucha dedicación y entusiasmo.

Este trabajo no hubiera sido posible sin la inmensa colaboración de Andrés Stoliar para superar los incovenientes que se presentaron al trabajar con el robot FenBot.

Quisieramos agradecer a todo el grupo de robótica del Departamento de Computación: Andrés Stoliar, Andrea Katz, Sergio Soria, por el aliento que nos dieron y la paciencia que nos tuvieron durante todo el trabajo.

Gracias a nuestras familias, que durante todos estos años nos brindaron todo su apoyo para seguir adelante con nuestros estudios. A Patricia, Daniel, Celeste y Rocío. A Ana, Héctor, Nadia y Mariel .

Esta tesis cierra una etapa en nuestras vidas como estudiantes universitarios. Etapa que estuvo marcada por nuestra militancia dentro del movimiento estudiantil. Queremos expresar nuestro más sentido reconocimiento a todos los compañeros de la Lista Unidad que no aceptan lo habitual como cosa natural y luchan por hacer posible lo necesario: la universidad del pueblo liberado. No podemos concluir entonces, sin antes agradecer a la Corriente Estudiantil Popular Antiimperialista y a la Juventud Comunista Revolucionaria, por brindarnos durante todos estos años una trinchera en la lucha por la liberación nacional y social de nuestro pueblo.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Un robot autónomo debe sensar su entorno y tomar decisiones mientras cumple una tarea específica, sin ningún tipo de intervención humana [1]. Los campos de aplicación más importantes incluyen tareas de reconocimiento, exploración, rescate, traslado y manipulación de objetos. El principal potencial para la utilización de robots autónomos en estas áreas radica en que pueden realizar una tarea en entornos hostiles o directamente imposibles para el trabajo humano.

Desde el punto de vista computacional y la teoría de control los robots autónomos plantean problemas interesantes que surgen principalmente del hecho de que la interacción con el mundo real es mucho más compleja que cualquier simulación: la cantidad de variables en juego es muy grande, los sensores son imprecisos, parciales y muchas veces contradictorios, los actuadores fallan o no responden de forma exacta. Desde el punto de vista de la psicología y la biología el campo de los robots autónomos brinda un marco ideal para estudiar aspectos relacionados con la inteligencia, el comportamiento animal y el proceso de adaptación y aprendizaje.

Una característica de los robots autónomos es su capacidad de adaptación. La adaptación no sólo consiste en tratar con los cambios de su entorno sino también la posibilidad de realizar acciones que introduzcan cambios que modifiquen su comportamiento posterior. Desde este punto de vista, podemos ver al aprendizaje como un caso extremo de adaptación.

Un modelo de aprendizaje que ha impactado en el campo de los robots autónomos es el Aprendizaje por Refuerzo. La definición de Aprendizaje por Refuerzo (AR) se ajusta exactamente a la descripción general del proceso de aprendizaje en robots: el robot debe incrementar la medida de desempeño, para una tarea asignada, a lo largo del tiempo [2].

En el AR, el robot debe aprender a asociar acciones a los estados percibidos del entorno mediante un mecanismo de premios y castigos. Se pueden distinguir

cuatro elementos en un sistema de aprendizaje por refuerzo: una política, una función de refuerzo, una función de valor y, opcionalmente, un modelo del entorno.

Una política define la forma en que un agente o robot va a comportarse en un determinado momento. Es un mapeo de los estados percibidos a las acciones que deben ser tomadas en esos estados. Una función de refuerzo define el objetivo en un problema de AR. La función de valor mapea cada estado percibido del entorno a un número que indica la deseabilidad de ese estado. Mientras que la función de refuerzo indica qué es bueno en un sentido inmediato, la función de valor indica qué es bueno a largo plazo, esto es, la esperanza de la suma de los refuerzos a recibir a partir del estado actual. Finalmente, un modelo del entorno permite conocer cuál es la probabilidad de pasar de un estado s a otro s' a partir de realizar la acción a [3].

Dentro de las técnicas de AR se destaca Q-Learning [4]. Un aspecto particular de esta técnica es que no requiere de un modelo explícito del entorno y esto la hace especialmente atractiva en el aprendizaje en robots. Mediante una estrategia de exploración, el robot experimenta directamente el entorno y aprende el valor de los refuerzos esperados al tomar una acción a en un determinado estado s, construyendo la función $Q(s, a)$. Luego, la política óptima se obtiene asociando a cada estado la acción que maximice la función Q.

La implementación computacional de las técnicas de AR requieren de una estructura de datos para almacenar la función de valor o la función de acción-valor Q. Típicamente esto se hace mediante la implementación de una tabla con una entrada para cada estado o para cada par de estado-acción. Esto limita la aplicación de AR a problemas con una pequeña cantidad de estados y acciones. El problema no es solamente la memoria necesaria para almacenar tablas grandes sino principalmente el tiempo y datos necesarios para llenarla de forma correcta. Para aprender correctamente la función de valor, las técnicas de AR requieren visitar todos los estados y acciones varias veces, lo cual no es posible desde un punto de vista práctico, si el cardinal del espacio de estados o acciones es muy grande. Esto ocurre cuando los espacios o acciones incluyen variables continuas o sensados complejos como imágenes visuales provenientes de una cámara de video, lo que sucede en una gran cantidad de problemas reales [5]. Ejemplos de este tipo de problemas son aquellos donde el robot debe detectar formas, colores, superficies o patrones, y donde otros sensores como sonar, sensor infrarrojo de proximidad, scaner láser o telémetro infrarrojo pueden resultar insuficientes. Una rama importante de problemas en los cuales una cámara de video suele ser el sensor más adecuado, son los de navegación. Por ejemplo, ciertos sistemas de navegación utilizando *landmarks* son imposibles de abordar sin la utilización de una cámara de video [7][8][9].

Existen formas de abordar el problema de la alta cardinalidad del espacio de estados. Las técnicas más usuales se basan en algún tipo de generalización,

como la utilización de la experiencia sobre un conjunto limitado del espacio de estados para generar una aproximación sobre un conjunto mucho más grande, o la aglomeración de estados "similares" en clusters[5]. Ambos enfoques no sólo buscan representaciones más compactas del espacio de estados o estado-acción sino también una forma de abordar el problema de la exploración exhaustiva del entorno.

Si el problema a tratar incluye una cámara de video como sensor principal, podemos utilizar transformaciones sobre las imágenes para obtener una nueva representación cuyo cardinal sea menor. En el área de Procesamiento de Imágenes existen numerosas técnicas que utilizan transformadas para analizar imágenes y que pueden ser utilizadas para reducir la cardinalidad del espacio de estados preservando la semántica de la imagen original [11]. Una muy utilizada es la transformada de Hough [23]. Esta transformada permite identificar formas en una imagen. Se basa en transformar los puntos de la imagen en un espacio de parámetros. La idea es encontrar curvas parametrizables como rectas, círculos y polinomiales.

Existen algunos trabajos que combinan el procesamiento de imágenes con técnicas de AR [12][13]. Si bien en estos trabajos se utiliza procesamiento de imágenes para componer parte del estado usado en el AR, se trata de problemas sumamente acotados que no tratan con alta cardinalidad de estados o acciones. Es decir, el espacio de estados no son las imágenes en sí (o una representación reducida de las mismas), sino que las imágenes son utilizadas para chequear su correspondencia con determinados estados objetivos. De acuerdo a nuestro conocimiento, no existen trabajos que utilicen transformadas como la de Hough para reducir el cardinal del espacio de imágenes manteniendo la semántica del espacio y permitiendo la aplicación de técnicas de AR sobre el mismo.

El objetivo de este trabajo es proponer un método que permita utilizar AR en problemas para los cuales una cámara de video es el sensor más adecuado, es decir, problemas en los cuales se tiene una percepción basada en imágenes capturadas del ambiente. Para mapear el espacio de imágenes visuales a una representación cuyo cardinal haga factible el uso de técnicas de AR como Q-Learning se propone usar la transformada de Hough. A diferencia de trabajos precedentes, no se pretende definir qué estados de esta nueva representación son relevantes. Esto último será llevado a cabo por la técnica de AR de acuerdo a la tarea a ser aprendida.

El resto de este manuscrito está organizado de la siguiente manera: el capítulo 2 presenta una introducción a los tópicos de AR que utilizaremos en el presente trabajo. Definiremos formalmente el modelo de AR, estudiaremos la propiedad de Markov, la ecuación de Bellman y el algoritmo de Q-Learning. Además, daremos una introducción a técnicas de exploración, al diseño de funciones de refuerzo y analizaremos las limitaciones que presenta AR en problemas con alta cardinalidad de estados. En el capítulo 3 se describen las

características principales de un sistema de visión. Estudiaremos en detalle la transformada de Hough para el caso lineal y presentaremos distintos métodos de detección de bordes, como forma de preprocesar las imágenes para poder aplicar el algortimo de Hough. El capítulo 4 describe en detalle el método propuesto para permitir la utilización de AR en problemas en los cuales se cuenta con una percepción del ambiente basada en visión. Además, analizaremos los distintos aspectos a tener en cuenta para poder aplicar este método. En el capítulo 5 se presentan los resultados experimentales en simulación para la tarea de recorrer un pasillo, utilizando el método propuesto en el capítulo anterior. En el capítulo 6 se presentan los resultados experimentales con el robot real FenBot para la misma tarea. Por último, en el capítulo 7 se encutran las conclusiones de este trabajo y las posibles líneas de investigación que permitirían continuar este trabajo en el futuro.

Capítulo 2

Aprendizaje por Refuerzo

En este capítulo se presenta una introducción a los tópicos de AR que utilizaremos en este trabajo. Definiremos formalmente el modelo de AR, estudiaremos la propiedad de Markov, la ecuación de Bellman y el algoritmo de Q-Learning. Además, daremos una introducción a técnicas de exploración, al diseño de funciones de refuerzo y analizaremos las limitaciones que presenta AR en problemas con alta cardinalidad de estados

2.1. Aprendizaje por Refuezo

En AR el robot debe aprender a asociar acciones a los estados percibidos del entorno mediante un mecanismo de premios y castigos. No se le dice al agente qué acciones tomar sino que debe experimentar para descubrir qué acciones llevan a maximizar el refuerzo a largo plazo. Por otro lado, en general las acciones no sólo tienen efecto en el refuerzo inmediato sino que también influyen en todos los refuerzos subsecuentes. Estas dos características - prueba y error y refuerzos demorados - son las dos más distintivas de las técnicas de AR.

2.2. Modelo de Aprendizaje por Refuerzo

En el modelo estandar de Aprendizaje por Refuerzo, un agente interactúa con el ambiente [1] en intervalos discretos de tiempo $t = 0, 1, 2, \ldots$. En cada instante de tiempo t, el agente percibe el entorno y construye una representación

[1] En general, esperamos que los ambientes sean no determinísticos, o sea, que tomar una acción en un estado dado pueda llevar a diferentes estados. Pero sí se asume que los ambientes son estacionarios, o sea, que la probabilidad de las transiciones de estados no cambian a lo largo del tiempo

del estado $s_t \in S$, donde S es el conjunto de posibles estados; y selecciona una acción $a \in A(s_t)$, donde $A(s_t)$ representa el conjunto de acciones posibles de realizar en el estado s_t. En el siguiente instante de tiempo, como parte de las consecuencias de la acción realizada, el agente recibe un refuerzo $r_{t+1} \in \Re$ y percibe un nuevo estado s_{t+1}. Esta interacción se esquematiza en la figura 2.1.

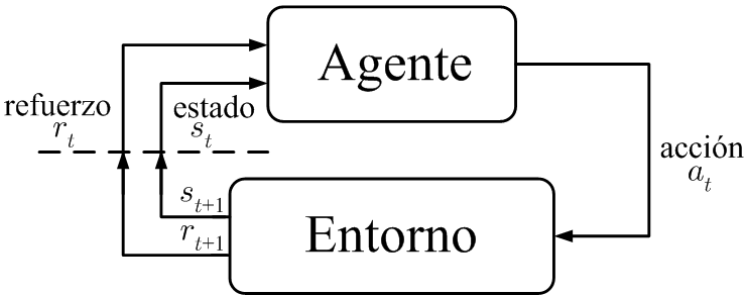


Figura 2.1: Esquema de interacción entre el agente y el entorno.

Para determinar las acciones a realizar el agente utiliza una política π, que relaciona cada estado s_t a una acción a_t. Los técnicas de AR tienen como objetivo obtener una política que maximice los refuerzos obtenidos a lo largo del tiempo.

2.3. Refuerzos, retorno y modelo de optimalidad

Hasta ahora hemos hablado informalmente sobre el objetivo del aprendizaje: maximizar los refuerzos que el agente recibe a lo largo del tiempo. ¿Cómo podemos definir esto formalmente? Si la secuencia de refuerzos recibidos luego del tiempo t es $r_{t+1}, r_{t+2}, r_{t+3}, ...$, ¿cuál es el aspecto preciso que queremos maximizar? En general, lo que queremos maximizar es el retorno esperado, donde el retorno R_t es una función de la secuencia de refuerzos futuros. En el caso más simple, el retorno R_t es la suma de los refuerzos:

$$R_t = r_{t+1} + r_{t+2} + ... + r_T$$

donde T es el instante de tiempo final. Esto tiene sentido en aplicaciones en las que es natural pensar en un paso final de tiempo, o sea, en aplicaciones que se pueden partir en subsecuencias o episodios.

Sin embargo, muchos casos no son divisibles en episodios identificables. Para esos casos, $T = \infty$ y por lo tanto definir el retorno como la suma directa de los refuerzos es imposible. Es por eso que el modelo de optimalidad más utilizado es el *infinite-horizon discount model* (modelo con descuento de horizonte

infinito), que tiene en cuenta todos los refuerzos a largo plazo, pero éstos son disminuidos geométricamente deacuerdo a un factor de descuento γ, (donde $0 \leq \gamma \leq 1$):

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

El factor de descuento γ determina el valor presente de los refuerzos futuros. Si $\gamma = 0$ o muy cercano a cero, el retorno es miope ya que sólo se preocupa de maximizar el refuerzo inmediato. Si γ se acerca a 1, el peso de los refuerzos futuros es mayor.

2.4. Propiedad de Markov

En AR el agente toma sus decisiones en función de una descripción del entorno que llamamos estado. Ciertamente el estado debería incluir mediciones inmediatas de los sensores, pero puede incluir mucho más que eso. Las representaciones de los estados pueden ser versiones altamente procesadas de los sensados originales, o pueden ser estructuras complejas construídas a lo largo del tiempo a partir de una secuencia de sensados.

Lo ideal sería tener un estado que sintetice los sensados pasados de forma compacta, pero de manera tal de tener toda la información necesaria. Una descripción del entorno que retiene toda la información relevante se dice de *Markov* o que cumple con la *propiedad de Markov*.

En el caso más general, r_{t+1} y s_{t+1} pueden depender de todo lo sucedido anteriormente. En este caso, la dinámica del sistema sólo puede definirse especificando la probabilidad completa:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., r_1, s_0, a_0\}. \tag{2.1}$$

Si los estados tienen la *propiedad de Markov* entonces r_{t+1} y s_{t+1} dependen solamente del estado y la acción en el tiempo t. En este caso, la dinámica del sistema se puede definir especificando solamente:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} \tag{2.2}$$

En otras palabras, una descripción del entorno tiene la propiedad de Markov si 2.1 es igual a 2.2. Esta propiedad es muy importante en AR porque se asume que las decisiones y valores son funciones sólo del estado actual.

2.5. Proceso de Decisión de Markov

Una tarea de AR que satisface la propiedad de Markov se llama Proceso de Decisión de Markov (MDP). Un MDP finito está definido por un conjunto finito de estados S, un conjunto finito de acciones A y la dinámica del ambiente determinada sólo por el paso actual. Dado un estado s y una acción a, la probabilidad de cada posible próximo estado s' está definida por:

$$P^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}.$$

Esto se llama probabilidad de transiciones. Similarmente, dado un estado actual s y acción a, junto con el próximo estado s', el valor esperado del próximo refuerzo es:

$$R^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}.$$

Esas cantidades $P^a_{ss'}$ y $R^a_{ss'}$ especifican completamente los aspectos más importantes de la dinámica de un MDP finito y reciben el nombre de dinámica del entorno (o del ambiente).

2.6. Función de Valor

Casi todos los algoritmos de AR están basados en estimar funciones de valor - funciones de estados (o pares de estado-acción) que estiman qué tan bueno es para el agente estar en un estado dado (o tomar la acción dada en un determinado estado). Esta noción de "bueno" está definida en términos de los futuros refuerzos que pueden ser esperados, o sea, de la esperanza del retorno. Por supuesto, los refuerzos esperados dependen de qué acciones tome el agente en el futuro, y por lo tanto las funciones de valor se definen según una política dada.

Una política $\pi(s, a)$ es una función de cada estado $s \in S$ y acción $a \in A(s)$ a la probabilidad de tomar la acción a en el estado s. El valor de un estado s dada una política π, denotado por $V^\pi(s)$ es el retorno esperado empezando desde s y siguiendo la política π.

$$V^\pi(s) = E_\pi \{R_t | s_t = s\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\}.$$

Similarmente, definimos el valor de tomar una acción a en el estado s siguiendo la política π, denotado por $Q^\pi(s,a)$, como el retorno esperado empezando por el estado s, tomando la acción a y siguiendo la política π a partir de allí.

$$Q^\pi(s,a) = E_\pi \{R_t | s_t = s, a_t = a\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right\}.$$

Una propiedad fundamental de las funciones de valor, muy usada en AR, es que satisface una relación recursiva particular:

$$\begin{aligned}
V^\pi(s) &= E_\pi\{R_t | s_t = s\} \\
&= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\} \\
&= E_\pi \left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s \right\} \\
&= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s' \right\} \right] \\
&= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right].
\end{aligned}$$

Esta ecuación, llamada *Ecuación de Bellman para* V^π, expresa la relación entre el valor de un estado y los valores de los estados sucesores. Empezando de un estado s, el agente puede tomar una de varias acciones. A partir de cada una de estas acciones, podemos llegar a uno de varios próximos estados s' y refuerzos r (ver figura 2.2). La ecuación de Bellman hace el promedio de todas las posibilidades, ponderando según la probabilidad de que ocurran. Nos dice que el valor del estado inicial debe ser igual al valor esperado del próximo estado más los refuerzos esperados a partir de ese estado.

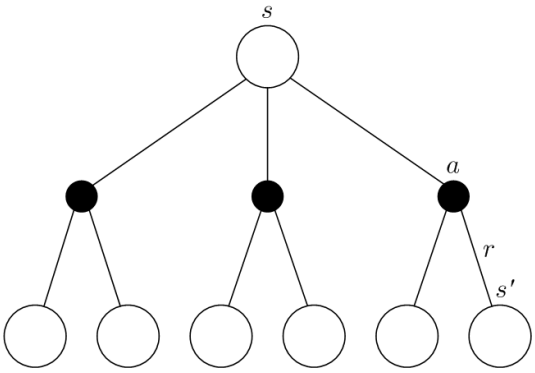


Figura 2.2: Diagrama de transición de estados.

2.7. Función de Valor Óptima

Podemos definir un orden parcial entre las políticas a partir de las funciones de valor:

$$\pi \geq \pi' \Leftrightarrow V^{\pi}(s) \geq V^{\pi'}(s) \quad \forall s \in S.$$

Siempre hay por lo menos una política que es mejor o igual que todas las demás. Esta política recibe el nombre de *política óptima*. La función de valor de estado óptima se define como la función de valor de esa política, o sea:

$$V^*(s) = \max_{\pi} V^{\pi}(s).$$

De la misma manera podemos definir la función de valor óptima de pares estado-acción:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a).$$

Para el par estado-acción (s,a), esta función da el retorno esperado de tomar la acción a en el estado s y luego seguir la política óptima, o sea:

$$Q^*(s,a) = E\left\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\right\}.$$

Como V^* es la función de valor para una política, debe satisfacer la ecuación de Bellman. Aparte, como es la función de valor óptima, puede escribirse en una forma especial sin referenciar ninguna política específica. Intuitivamente, esta ecuación expresa que el valor de un estado bajo la política óptima debe ser igual al retorno esperado para la mejor acción de ese estado:

$$
\begin{aligned}
V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s,a) = \max_{a \in A(s)} Q^*(s,a) \\
&= \max_a E\left\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\right\} \\
&= \max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^*(s')\right]
\end{aligned}
$$

y para Q^*:

$$
\begin{aligned}
Q^*(s,a) &= E\left\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a\right\} \qquad (2.3) \\
&= \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a'} Q^*(s', a')\right]. \qquad (2.4)
\end{aligned}
$$

Las ecuaciones de optimalidad de Bellman son, en realidad, un sistema de ecuaciones, una por cada uno de los n estados, con n incógnitas. Por lo tanto, si conocemos la dinámica del entorno (P y R), se puede encontrar la política óptima resolviendo ese sistema de ecuaciones. En la mayoría de los casos es imposible resolver el sistema de ecuaciones de Bellman para encontrar la política óptima, ya sea porque no conocemos la dinámica del entorno, no se cumple estrictamente la propiedad de Markov o no poseemos la capacidad computacional suficiente para resolver ese sistema. Sin embargo, estas ecuaciones sirven para inspirar muchos métodos de AR.

2.8. Q-Learning

El algoritmo más difundido dentro de las técnicas de AR es Q-Learning que fue originalmente propuesto por Watkins [4]. Se basa en aproximar directamente Q^* utilizando un método basado en la ecuación de Bellman de optimalidad 2.3.

Mirando esa ecuación, podemos pensar en la siguiente similitud:

$$
Q^*(s,a) \approx r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a')
$$

Ahora bien, como no conocemos $Q^*(s_{t+1}, a')$ y aparte no es cierta la igualdad en el caso general[2], es necesario que la actualización de $Q(s,a)$ no sea directamente lo anterior. Por eso se introduce el coeficiente de aprendizaje α ($0 < \alpha < 1$) para actualizar el valor de $Q(s,a)$:

[2]Sí es cierta si el ambiente es determinístico.

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'))$$

El algoritmo de Q-Learning es libre de modelo. Esto es, no necesita conocer la dinámica del ambiente, sólo necesita explorarlo. Su convergencia no depende de una política especial para definir, durante el aprendizaje, qué acción tomar en cada estado. Asumiendo que a lo largo del tiempo los pares estado-acción siguen siendo visitados, está demostrado que Q_t converge a Q^* [6]. Estas propiedades y su simplicidad hacen que Q-Learning sea uno de los algoritmos de AR más utilizados. A continuación se muestra el algoritmo de Q-Learning:

```
Inicializar Q(s,a) en forma arbitraria.
Observar el estado s.
Repetir:
    Elegir una acción a siguiendo alguna política.
    Ejecutar la acción a.
    Observar el refuerzo r y el nuevo estado s'.
    Q(s,a) ← (1- α ) Q(s,a) + α (r + γ max_a' Q(s',a')).
    s ← s'.
hasta que se satisfaga algún criterio de parada.
```

Figura 2.3: Algoritmo de Q-Learning.

2.9. Exploración eficiente

Una pregunta queda pendiende: ¿cómo elige el agente qué acción tomar en cada estado mientras está aprendiendo? En otras palabras, ¿qué política sigue el agente durante el aprendizaje? Cuando un agente aprende en un ambiente desconocido, dos objetivos contrapuestos deben ser combinados. Por un lado, para obtener mayor refuerzo, el agente debe elegir acciones que ya ha ejecutado y que le han dado refuerzos positivos en el pasado. Pero por el otro lado, para descubrir esas acciones tiene que tomar acciones que no ha intentado antes. Es decir, el agente debe *explotar* lo que ya sabe para obtener refuerzos, pero también debe *explorar* para tomar mejores decisiones en el futuro. Todas las estrategias de exploración tienen en cuenta este dilema, ya que la pura exploración o la pura explotación no son buenas estrategias para resolver un problema de AR. [14]

Según Thrun [14] [15], las estrategias de exploración pueden dividirse en dos grandes grupos: exploración no dirigida y exploración dirigida. Mientras que el primer grupo no utiliza ningún conocimiento específico de la exploración y asegura la exploración utilizando azar en la selección de las acciones, las

técnicas dirigidas confían en el conocimiento acerca del proceso de aprendizaje en sí, permitiendo la exploración de una manera más inteligente.

Entre las primeras se encuentra, por ejemplo, la exploración puramente random (se elige siempre una acción al azar). Una mejora sobre esta política es la técnica ϵ-greedy. En este caso, dado un ϵ entre 0 y 1, con probabilidad ϵ se elige una acción al azar, y con probablidad $1 - \epsilon$ se elige la mejor acción hasta el momento. En Q-Learning se define como mejor acción aquella que para el tiempo t y el estado s maximice la función $Q_t(s, a)$, es decir, $a^* = max_a Q_t(s, a)$. Comenzando con un ϵ cercano a 1 y disminuyendo gradualmente el ϵ podemos pasar de manera suave de la pura exploración a la pura explotación.

Entre las técnicas de exploración dirigidas se destaca la técnica counter-based [15], cuya idea es explorar los estados menos visitados. O sea, elegir en un estado dado la acción que nos lleva al estado menos explorado. Para cada estado s se define un contador $c(s)$ para saber cuántas veces se ha visitado ese estado. Las acciones son elegidas evaluando una combinación lineal entre el término de explotación y de exploración:

$$eval_c(a) = \alpha f(a) + \frac{c(s)}{E[c|s, a]}$$

donde $f(a)$ es la utilidad estimada de la acción a, o sea, qué tan bueno es tomar la acción a en el estado actual, basándonos en lo que conocemos hasta el momento. $E[c|s, a] = \sum_s P^a_{ss'} c(s')$ es el valor esperado del contador del estado obtenido al aplicar la acción a en el estado s. Y α es una constante que pesa la exploración versus la explotación.

En esta técnica no hay azar: siempre se elige la acción que maximiza $eval_c$ de manera determinística. Un claro problema de esta técnica es que requiere conocer $P^a_{ss'}$.

La estrategia de exploración que se elija es sumamente importante, ya que para aprender correctamente la función de valor, las técnicas de AR (y en particular Q-Learning) requieren visitar todos los estados y acciones varias veces. Esto puede ser un problema si tenemos un espacio de estados cuyo cardinal es muy grande. La mayor parte de los estados encontrados durante la exploración nunca habrán sido visitados anteriormente y por ende no se aprenderá correctamente qué acción tomar. En [15] Thrun muestra que las técnicas de exploración dirigida previenen, en muchos casos, tiempos de exploración exponenciales en la cantidad de estados. También en [15] Thrun muestra que en problemas con espacio de estados muy grandes las técnicas de exploración no dirigidas a veces no llevan a aprender políticas aceptables. Es por esto que las técnicas de exploración dirigidas son útiles en todos los problemas de AR, pero son imprescindibles a la hora de tratar con problemas con alta cardinalidad de estados.

2.10. Funciones de Refuerzo

El éxito de una aplicación de AR se debe en gran medida al diseño de una función de refuerzo apropiada. Todavía no se ha desarrollado un método para el diseño de las funciones de refuerzo. Esta tarea crítica y generalmente subestimada es dejada a la habilidad de los diseñadores de AR [10][16].

En general las funciones de refuerzo tratan con dos tipos de refuerzos. Se pueden definir *refuerzos inmediatos* luego de la ejecución de cada acción si es posible estimar cuánto aporta cada acción a alcanzar el objetivo. Este tipo de funciones de refuerzo dan mucha información y, cuando son posibles, parecen más atractivas que aquellas que ofrecen refuerzo sólo en determinadas situaciones, cuando se alcanza algún objetivo o sub-objetivo. Estos refuerzos reciben el nombre de *refuerzos demorados*[16].

Los refuerzos inmediatos pueden introducir algunos problemas. En general este tipo de funciones requieren mucho conocimiento del entorno *a priori* [17], lo que en principio puede introducir un bias en el aprendizaje en direcciones no deseadas.

Los refuerzos demorados están muy relacionados con la forma en la que se explicitan los objetivos en AR. Un problema típico con el refuerzo demorado es su distribución sobre el espacio de estados. Si el refuerzo es muy poco probable, entonces los algoritmos de AR funcionan mal, ya que durante la mayor parte del aprendizaje el agente no sabe si las acciones que está tomando lo acercan hacia el objetivo. Es por esto que no son adecuados por sí solos para situaciones con muchos estados ni para trabajar en un robot real.

En [17] se reconoce el problema de los refuerzos inmediatos, pero se pone de relieve la importancia de tener refuerzos contínuos para acelerar el aprendizaje. Se propone incorporar a las funciones de refuerzo estimadores de progreso para introducir refuerzos inmediatos. Los estimadores de progreso dan idea acerca de mejoría con respecto a uno o más objetivos puntuales, y son una manera de acelerar el aprendizaje incorporando conocimiento sobre el entorno a la función de refuerzo.

Finalmente, la función de refuerzo puede utilizarse también para introducir información sobre el entorno. Por ejemplo, en un robot móvil que debe evitar obstáculos, podemos incluir refuerzos negativos al chocar contra un objeto.

2.11. AR con alta cardinalidad de estados

La implementación computacional de las técnicas de AR requieren de una estructura de datos para almacenar la función de valor o la función de acción-valor Q. Típicamente esto se hace mediante la implementación de una tabla

con una entrada para cada estado o para cada par de estado-acción. Esto limita la aplicación de AR a problemas con una pequeña cantidad de estados y acciones. El problema no es solamente la memoria necesaria para almacenar tablas grandes sino principalmente el tiempo y datos necesarios para llenarla de forma correcta. Para aprender correctamente la función de valor, las técnicas de AR requieren visitar todos los estados y acciones varias veces, lo cual no es posible desde un punto de vista práctico, si el cardinal del espacio de estados o acciones es muy grande. Esto ocurre cuando los espacios o acciones incluyen variables continuas o sensados complejos como imágenes visuales provenientes de una cámara de video [5].

Existen formas de abordar el problema de la alta cardinalidad del espacio de estados. Las técnicas más usuales se basan en algún tipo de generalización. El conjunto de técnicas de generalización existentes puede dividirse en dos grandes grupos. El primero de ellos plantea romper con la representación tabular definida anteriormente para Q, introduciendo un problema de aproximación de funciones a partir de ejemplos. La generalización desde ejemplos es un tema que ha sido ampliamente estudiado en otros dominios fuera de AR. Este tipo de generalización es a menudo denominada aproximación de funciones, dado que toma ejemplos de una función que se desea aprender (por ejemplo, la función Q) e intenta generalizar esos ejemplos para construir una aproximación completa de dicha función. Uno de los métodos más habituales para realizar esta estimación son los métodos no lineales de descenso de gradiente, que suelen incluir el uso de redes neuronales como aproximadores.

Existen muchos trabajos en esta línea, entre los que destacan RBF (Radial Basis Function) [18] y redes de retro-propagación (*backropagation*) para Q-Learning en [19]. No obstante, esta forma de aproximación de Q introduce grandes problemas de convergencia, dado que al realizar el aprendizaje a partir de ejemplos de entrenamiento limitados, los pequeños errores de clasificación que se producen al inicio del aprendizaje en determinadas zonas del espacio de estados son propagados al resto del espacio, impidiendo que la función sea aproximada correctamente.

El segundo tipo de generalización mantiene la representación tabular, pero reduce el espacio de estados original mediante la aglomeración de estados "similares" en clusters, de forma que se puede hacer una traducción de cualquier estado del conjunto inicial a un estado del conjunto reducido. Se trata también, por tanto, de una aproximación de funciones, pero en este caso, lineal [3]. Los principales problemas que estas técnicas introducen son, por un lado, definir cuál es el número de estados adecuado para la nueva representación, y por otro lado, definir los límites entre esos estados.

Si el problema a tratar incluye una cámara de video como sensor principal, podemos utilizar transformaciones sobre las imágenes para obtener una nueva representación cuyo cardinal sea menor. En el área de Procesamiento

de Imágenes existen numerosas técnicas que utilizan transformadas para analizar imágenes y que pueden ser utilizadas para reducir la cardinalidad del espacio de estados preservando la semántica de la imagen original [11]. Por lo tanto, si el problema de AR a tratar incluye una percepción del entorno basada en visión, podemos intentar utilizar este enfoque para tratar con la alta cardinalidad de estados.

Capítulo 3

Visión

En este capítulo daremos una introducción a algunos tópicos de visión y procesamiento de imágenes que utilizaremos en este trabajo. Describiremos las características principales de un sistema de visión. Estudiaremos en detalle la transformada de Hough para el caso lineal y presentaremos distintos métodos de detección de bordes, como forma de preprocesar las imágenes para poder aplicar el algortimo de Hough.

3.1. Sistema de visión

Desde el punto de vista de la fisiología animal, la visión es un sentido que proporciona al sistema nervioso de una vía única de comunicación con el medio exterior. Constantemente se captan señales del ambiente que al interaccionar con la organización del sistema nervioso le proporcionan a cada animal su acervo de conocimiento. Para el reconocimiento de patrones o para el control de la locomoción, los animales necesitan de un sistema óptico que les permita formar imágenes. Las redes neuronales sensoriales son las encargadas de generar respuestas a los estímulos visuales para realizar un comportamiento adecuado. No sólo refinan y clasifican la información que es accesible al animal, sino que pueden magnificar, ampliar, añadir, restar e incluso reconfigurar por completo el patrón original de la señal sensorial. El procesamiento visual involucra una cadena de procesos de abstracción y acentuación de las propiedades y características del estímulo visual. A lo largo del desarrollo del individuo, el este procesamiento visual se va educando mediante nuevas conexiones neuronales. Desde este punto de vista, podemos decir que el sentido de visión requiere de un proceso de aprendizaje [20].

Desde el punto de vista de la robótica, la visión es un sensor más que los robots pueden utilizar para inferir el estado de su entorno. Es un sensor complejo que entrega un flujo contínuo de imágenes. No es sencillo extraer

de ese flujo información directamente útil para el robot. Un sistema de visión robótico debe analizar las imágenes y producir un descripción del entorno que capture los aspectos que permitan llevar adelante una determinada tarea. Podemos considerar al sistema de visión como la parte del robot que está a cargo del sensado, mientras otras partes están dedicadas a decidir qué acción tomar y a la ejecución de dicha acción. Si el robot cuenta con otros sensores, las señales de los mismos deben combinarse con el sistema de visión para completar la descripción del entorno.

El *input* de un sistema de visión robótico es un flujo contínuo de imágenes sensadas con una cámara de video, mientras que el *output* es una representación que debe satisfacer dos criterios:

1. Debe mantener una relación con lo que está siendo capturado por las imágenes.
2. Debe contener la información necesaria para realizar la tarea dada.

El primer criterio nos asegura que la representación depende en algún sentido del *input* visual. El segundo criterio asegura que la información provista es apropiada. Todo objeto real no tiene una única descripción. Podemos tener distintos niveles de detalle y muchos puntos de vista. Es imposible decribir un objeto de forma completa, acabada. Afortunadamente, podemos esquivar este problema filosófico considerando la tarea para la cual la representación es concebida. No queremos sólo una representación que describa lo que es capturado por la imagen, sino una que permita tomar la acción correcta para la tarea que debe realizar el robot [22].

3.2. Transformada de Hough

Para extraer información de una imagen digital, es sumamente útil poder encontrar formas simples como rectas, círculos o polinomiales en la imagen. Esto es lo que la transformada de Hough [23] nos permite hacer.

El caso más simple es la transformada lineal de Hough. En este caso queremos detectar puntos colineales. Para eso tenemos que caracterizar las rectas que aparecen en la imagen. Podemos describir una recta en la imagen a partir de todos los puntos (x, y) que satisfacen la ecuación:

$$y = mx + b. \tag{3.1}$$

La caracterización de la recta no viene dada por x o y, sino por la pendiente m y la ordenada al origen b. Desde este punto de vista, una línea recta

puede ser representada como un punto (b_0, m_0) en el espacio de parámetros mb. Sin embargo, no es del todo apropiado utilizar la pendiente y la ordenada al origen como parámetros para hacer aplicaciones computacionales ya que dichos parámetros no están acotados. A medida que una recta es cada vez más vertical, las magnitudes de m y b crecen hacia el infinito. Para un propósito computacional, es mejor parametrizar las rectas con otros dos parámetros que no tengan este problema [24]. Para eso utilizaremos un parámetro ρ que representa la distancia mínima entre la recta y el origen de coordenadas y otro parámetro θ que representa el ángulo desde el origen hasta el vector que une el punto más cercano de la recta con el orgien. Esta forma de caracterizar una recta recibe el nombre de parametrización normal (o forma normal) de una recta (ver figura 3.1). Usando esta parametrización, la ecuación de la recta puede reescribirse como:

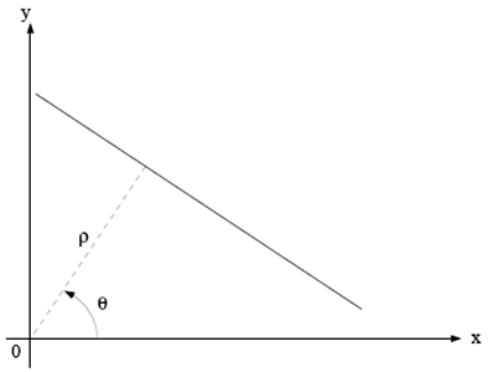


Figura 3.1: Representación normal de una recta.

$$\rho = xcos(\theta) + ysen(\theta). \tag{3.2}$$

Es posible seguir asociando a cada recta de la imagen una tupla (ρ, θ) de forma única si θ pertenece al intervalo $[0, \pi)$ y ρ pertenece al intervalo $[\frac{-diag}{2}, \frac{diag}{2}]$ donde $diag$ es la diagonal de la imagen. El espacio de parámetros $\rho\theta$ se denimina espacio de Hough.

Por un sólo punto del plano pasan infinitas rectas. Si ese punto tiene coordenadas (x_0, y_0) en el plano xy de la imagen, todas las rectas que pasan por ese punto tienen que satisfacer la siguiente ecuación:

$$\rho(\theta) = x_0 cos(\theta) + y_0 sen(\theta). \tag{3.3}$$

Esto corresponde a una curva sinusoidal en el plano $\rho\theta$ que es única para el punto (x_0, y_0). Si tenemos otro punto (x_1, y_1), entonces vamos a tener otra

sinusoidal en el plano $\rho\theta$. Si estas dos sinusoidales se cortan (en el plano $\rho\theta$) en el punto (ρ_0, θ_0), entonces existe una recta que pasa por (x_0, y_0) y (x_1, y_1) (en el plano xy) y esa recta es:

$$\rho_0 = xcos(\theta_0) + ysen(\theta_0). \tag{3.4}$$

Generalizando, a un grupo de puntos que forman una recta (en el plano xy) le van a corresponder un grupo de sinusoidales (en el plano $\rho\theta$) que se van a cortar en los parámetros de esa recta. Luego, el problema de detectar puntos colineales puede ser traducido al problema de encontrar los puntos donde se cortan sus respectivas sinusoidales.

3.2.1. Algoritmo

Lo primero que debemos tener en cuenta para implementar la transformada de Hough es cuáles pixels de la imagen van a ser candidatos a pertenecer a una recta. Sabemos que las rectas se encuentran en las fronteras entre dos regiones discontínuas (ej. con nivel de gris relativamente diferente) de la imagen. Por eso, para que un pixel sea candidato a estar en una recta debe pertenecer a una de estas fronteras o bordes de la imagen. De lo anterior, se desprende que el primer paso para aplicar la transformada de Hough es la detección de bordes en la imagen original. En la sección 3.3 analizaremos esto en detalle.

En segundo lugar, debemos escoger una determinada granularidad para los parámtros ρ y θ (ej 1 grado, un pixel). De esta forma, vamos a subdivir el espacio de parámetros en celdas acumuladoras. Inicialmente se ponen todos los acumuladores en cero. Para cada punto (x, y) de la imagen que pertenece a un borde (o sea, es candidato a estar en una recta), iteramos a través de los angulos definidos para esa granularidad. Para cada ángulo θ_j, obtenemos ρ_i resolviendo la ecuación

$$\rho_i = x.cos(\theta_j) + y.sin(\theta_j) \tag{3.5}$$

e incrementamos el valor para la celda acumuladora (ρ_i, θ_j) correspondiente. Al final el proceso, el valor en cada celda acumuladora (ρ_i, θ_j) representa la cantidad de puntos en el plano xy que se encuentran en la recta $\rho_i = x.cos(\theta_j) + y.sin(\theta_j)$. La precisión en la colinealidad de estos puntos depende del cardinal de la partición del espacio de parámetros. Finalmente se obtienen los parámetros de las rectas buscando los valores máximos en las celdas acumuladoras.

Si subdividimos el ángulo θ en k celdas, para cada punto (x, y) de la imagen que pertenece a un borde tenemos k valores de ρ. Si la imagen tiene n puntos,

y $m \leq n$ que pertenecen a un borde, el orden del algoritmo es $\mathcal{O}(n + mk)$. Luego podemos decir que esta implementación de la transformada de Hough es lineal respecto a la cantidad de pixels de la imagen.

3.3. Detección de Bordes

La detección de bordes en una imagen reduce significativamente la cantidad de información y filtra la información innecesaria, preservando las características fundamentales de la imagen. De allí que los métodos para detectar bordes sean una herramienta muy poderosa para el preprocesamiento de imágenes. Podemos definir como un borde a los pixels donde la intensidad de la imagen cambia de forma abrupta. Si consideramos una función de intensidad de la imagen, entonces lo que buscamos son saltos en dicha función.

La idea básica detrás de cualquier detector de bordes es el cálculo de un operador local de derivación. En la figura 3.2 se puede ver este concepto. En la parte izquierda se puede ver una imagen de una banda clara sobre un fondo oscuro, el perfil a lo largo de una línea horizontal y la primera y segunda derivada de dicho perfil. Se puede observar que el perfil del borde se ha modelado como una discontinuidad suave. Esto tiene en cuenta el hecho de que en las imágenes reales los bordes están ligeramente desenfocados. Como se puede observar en la figura 3.2 la primera derivada es positiva para cambio a nivel de gris más claro, negativa en caso contrario y cero en aquellas zonas con nivel de gris uniforme. La segunda derivada presenta valor positivo en la zona oscura de cada borde, valor negativo en la zona clara de cada borde y valor cero en las zonas de valor de gris constante y justo en la posición de los bordes. El valor de la magnitud de la primera derivada nos sirve para detectar la presencia de bordes, mientras que el signo de la segunda derivada nos indica si el pixel pertenece a la zona clara o a la zona oscura. Además, la segunda derivada presenta siempre un cero en el punto medio de la transición. Esto puede ser muy útil para localizar bordes en una imagen.

Aunque lo que llevamos dicho hasta aquí se refiere a perfiles unidimensionales, la extensión a dos dimensiones es inmediata. Simplemente se define el perfil en la dirección perpendicular a la dirección del borde y la interpretación anterior seguirá siendo válida. La primera derivada en cualquier punto de la imagen vendrá dada por la magnitud del gradiente, mientras que la segunda derivada vendrá dada por el operador Laplaciano. La mayoría de los métodos de detección de bordes pueden dividirse en dos grandes grupos: el método del gradiente detecta los bordes buscando los máximos y mínimos en la primera derivada de la imagen, mientras que el método del Laplaciano busca los ceros de la segunda derivada. Para este trabajo nos focalizaremos en el primer grupo.

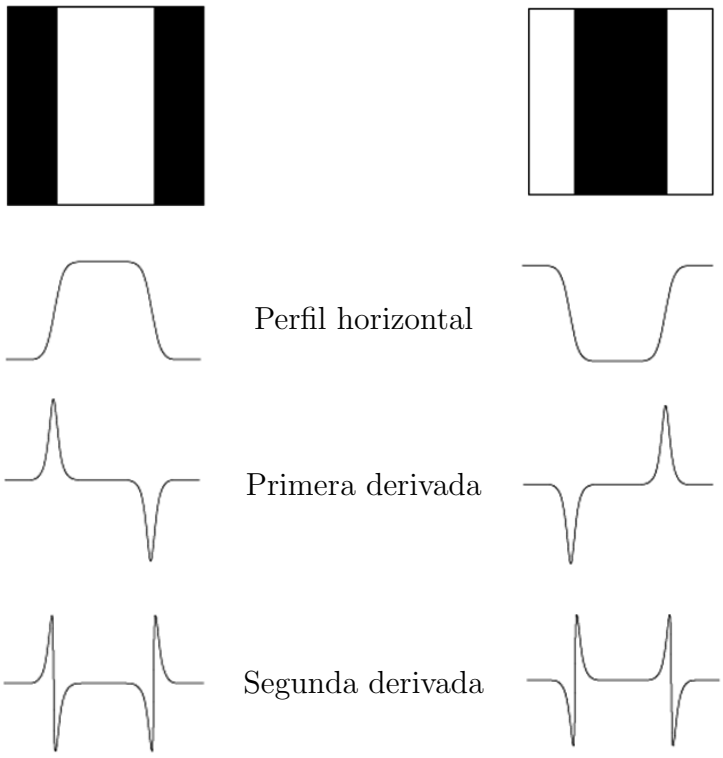


Figura 3.2: Detección de bordes empleando operadores de derivación. La primera derivada es positiva para cambio a nivel de gris más claro, negativa en caso contrario y cero en zonas con nivel de gris uniforme. La segunda derivada tiene un cero en la posición de cada borde.

3.3.1. Operadores de gradiente

El gradiente de una imagen $I(x, y)$ en la posición (x, y) viene dado por el vector

$$\nabla I = \begin{bmatrix} I_x \\ \\ I_y \end{bmatrix} = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \\ \frac{\partial I}{\partial y} \end{bmatrix}. \tag{3.6}$$

El vector gradiente siempre apunta en la dirección de la máxima variación de la imagen I en el punto (x, y). Para los métodos de gradiente nos interesa conocer la magnitud de este vector, denominado simplemente como gradiente de la imagen, denotado por $||\nabla I||$ y dado por

$$||\nabla I|| = \sqrt{I_x^2 + I_y^2}. \tag{3.7}$$

Esta cantidad representa la variación de la imagen $I(x, y)$ por unidad de distancia en la dirección del vector ∇I. En general, el gradiente de la imagen se suele aproximar mediante la expresión

$$||\nabla I|| \approx |I_x| + |I_y|, \tag{3.8}$$

que es mucho más simple de implementar computacionalmente [25].

Para el cálculo del gradiente de la imagen en cada pixel tenemos que obtener las derivadas parciales. Las derivadas se pueden implementar digitalmente de varias formas. Una forma es mediante máscaras de tamaño 3×3 o incluso más grandes. La ventaja de utilizar máscaras grandes es que los errores producidos por efectos del ruido son reducidos mediante promedios locales tomados en los puntos en donde se superpone la máscara. Por otro lado, las máscaras normalmente tienen tamaños impares, de forma que los operadores se encuentran centrados sobre el pixel en el que se calcula el gradiente. El requisito básico de un operador de gradiente es que la suma de los coeficientes de la máscara sea nula, para que la derivada de una zona uniforme de la imagen sea cero.

Los operadores de gradiente común (o gradiente ortogonal) encuentran bordes horizontales y verticales. Estos operadores trabajan mediante convolución. Los operadores de Roberts, Prewitt, Sobel y Frei-Chen son operadores dobles o de dos etapas. La detección de bordes se realiza en dos pasos. En el primero se buscan bordes horizontales utilizando la máscara que corresponde a la derivada parcial en x. En el segundo paso se buscan los bordes verticales utilizando la máscara que corresponde a la derivada parcial en y. Finalmente, se suman ambos para obtener el gradiente de la imagen en cada pixel. En la figura 3.3 se pueden ver los operadores de Roberts, Prewitt, Sobel y Frei-Chen para determinar bordes horizontales (izquierda) y verticales (derecha).

Los operadores de Sobel y de Frei-Chen tienen la ventaja de que proporcionan un suavizado además del efecto de derivación. Ya que la derivación acentúa el ruido, el efecto de suavizado es particularmente interesante, puesto que elimina parte del ruido. Además, la máscara de Sobel utiliza números naturales mientras que Frei-Chen utiliza números irracionales, por lo que el primero resulta numéricamente más estable. Es por esto que para el presente trabajo se utilizó el operador de Sobel.

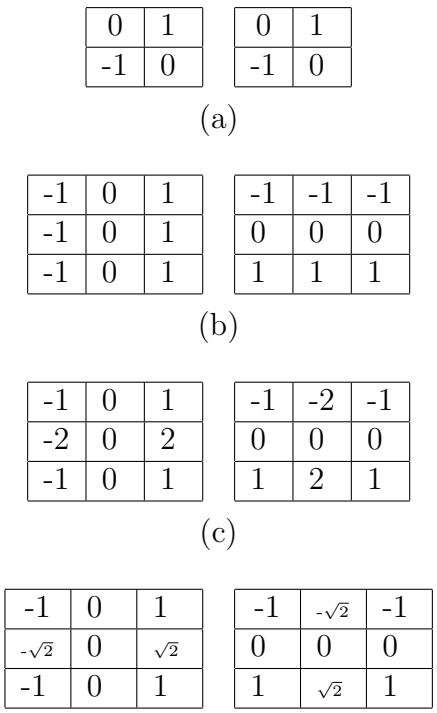


Figura 3.3: Operadores de derivación: (a) de Roberts, (b) de Prewitt, (c) de Sobel y (d) de Frei-Chen.

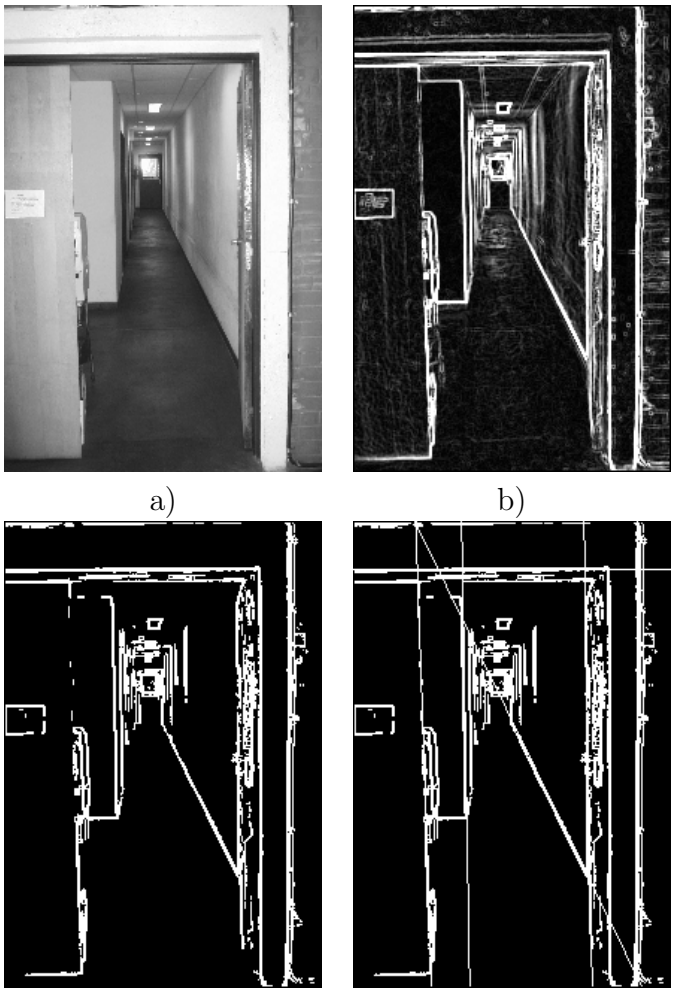
3.3.2. Umbralización

Una vez que aplicamos el operador de gradiente, tenemos que utilizar un método de umbralización para determinar cuáles pixels son considerados parte de un borde. La forma más sencilla es fijar un umbral y considerar borde a todo pixel cuya intensidad supere dicho umbral. Si fijamos un umbral bajo, más bordes van a ser detectados, pero el resultado se vuelve crecientemente más susceptible al ruido y a seleccionar características irrelevantes de la imagen. Contrariamente, un umbral alto puede que no detecte bordes suaves o que se van atenuando.

Una forma de tratar este problema es mediante la *umbralización con histéresis*. Este método usa múltiples umbrales. Se comienza utilizando un umbral superior para encontrar el inicio de un borde. Una vez que tenemos este punto de partida, se recorre la línea de borde marcando cada pixel adyacente al anterior recorrido cuya intensidad está por encima del umbral inferior. Dejamos de marcar el borde sólo cuando no hay pixels adyacentes con intensidad superior al umbral inferior. De esta forma podemos seguir bordes que se atenúan, sin tener que marcar el ruido de la imagen como bordes.

Este enfoque para tratar el problema de la umbralización asume que los bordes que buscamos en la imagen son líneas contínuas. Como el objetivo final de nuestro procesamiento de imágenes es la detección de rectas, nos interesa detectar bordes en la imagen que sean candidatos a ser rectas. Luego, la hipótesis del método de umbralización con histéresis se satisface.

En la figura 3.4 podemos ver el resultado de aplicar el operador de Sobel con umbralización con histéresis a una imagen de un pasillo.



a) b)

c) d)

Figura 3.4: a) Imagen original. Pasillo de oficinas del Departamento de Computación, FCEyN, UBA. b) Imagen obtenida al aplicar el operador de Sobel. c) Imagen obtenida al aplicar umbralización con histéresis. d) 6 rectas obtenidas a partir de la imagen c) con el algoritmo de Hough.

Capítulo 4

Aprendizaje por Refuerzo con percepción basada en visión

En este capítulo se presenta el método propuesto para permitir la utilización de AR en problemas donde se cuenta con una percepción del entorno basada en visión. Primero describiremos antecedentes de trabajos relacionados. Luego describiremos en detalle el método propuesto y analizaremos los distintos aspectos a tener en cuenta para su utilización.

4.1. Antecedentes

Existen algunos trabajos que combinan el procesamiento de imágenes con técnicas de AR. En [12] un robot debe posicionar su brazo en una determinada configuración relativa a un objeto conocido dispuesto sobre una mesa. Se analizan imágenes para formar un estado conformado por dos ángulos y una distancia. En [13] el robot debe aprender a tomar un objeto circular de tamaño conocido. Para ello debe mover un brazo que posee una cámara hasta que el objeto se encuentre en el centro de la imagen y su radio se perciba con un determinado tamaño. En este trabajo, los estados del robot son ciertos ángulos de su brazo, y se utilizan las imágenes para saber si se ha alcanzado el objetivo.

Si bien en estos trabajos se utiliza procesamiento de imágenes para componer parte del estado usado en el AR, se trata de problemas sumamente acotados que no tratan con alta cardinalidad de estados o acciones. Es decir, el espacio de estados no son las imágenes en sí (o una representación reducida de las mismas), sino que las imágenes son utilizadas para chequear su correspondencia con determinados estados objetivos. De acuerdo a nuestro conocimiento, no existen trabajos que utilicen procesamiento de imágenes para reducir el cardinal del espacio de imágenes manteniendo la semántica del

espacio y permitiendo la aplicación de técnicas de AR sobre el mismo.

4.2. Método Propuesto

La idea del método propuesto es utilizar la transformada de Hough para mapear el espacio de imágenes visuales a una representación cuyo cardinal haga factible el uso de técnicas de AR como Q-Learning. En otras palabras, nuestra propuesta es utilizar el espacio de Hough como espacio de estados para la técnica de Q-Learning.

Nuestra hipótesis de trabajo es que una representación basada en una cantidad pequeña de rectas inferidas a partir de las imágenes sensadas contiene la información necesaria del entorno y reduce el cardinal de estados lo suficiente, como para poder utilizar el algoritmo de Q-Learning.

Como puede observarse en el algortimo general de Q-Learning hay tres elementos que deben precisarse de acuerdo a las característica del robot (qué tipo de sensores y actuadores posee), el entorno y la tarea específica que debe llevar adelante el agente. A saber:

1. Cómo se percibe el estado $s \in S$. Para eso debemos elegir una representación del entorno que defina el espacio de estados S y proponer un método que permita contruir esa representación a partir del sensado del entorno.

2. Qué política π se utiliza durante el aprendizaje. Es decir, cómo elige el agente qué acción a tomar en cada estado s mientras está aprendiendo.

3. Cuál es el refuerzo r obtenido al ejecutar la acción a en el estado s.

En las próximas secciones describiremos en detalle cada uno de estos puntos.

4.3. Representación del entorno

Como dijimos anteriormente, para representar el entorno utilizaremos las rectas derivadas a partir de las imágenes capturadas. En la sección 3.2 vimos que se puede utilizar la transformada lineal de Hough para obtener estas rectas. Sin embargo, el espacio de todas las rectas de una imágen sigue siendo un espacio demasiado grande para tratar con técnicas de Aprendizaje por Refuerzo. Si tenemos t valores posibles de θ y r valores posibles de ρ, entonces hay $r \times t$ posibles rectas en la imágen. Si tomamos imágenes con n rectas,

entonces podemos distinguir aproximadamente $\frac{(r \times t)^n}{n!}$ imágenes diferentes (ver 4.3.2 para el desarrollo matemático exacto de esta cota).

En nuestro caso particular de estudio vamos a trabajar con imágenes de 640×480 pixels. Si tomamos una granularidad para la transformada de Hough de un ρ por pixel y un θ por grado, tenemos 800 valores distintos de ρ y 180 valores distintos de θ [1]. Esto nos da $800 \times 180 = 144{,}000$ rectas posibles en una imágen. Si queremos representar el entorno sensado con 10 rectas, tenemos aproximadamente $\frac{144{,}000^{10}}{10!}$ posibles estados. Esto es un tamaño de espacio de estados claramente inmanejable para AR (¡144,000 ya es un tamaño inmanejable!).

4.3.1. Clasificación de situaciones

Necesitamos entonces agrupar situaciones percibidas diferentes del entorno a un mismo *cluster*, para reducir el cardinal de este espacio. Como vimos anteriormente, lo que determina el cardinal del espacio de estados es la cantidad de valores que pueden tomar los parámetros ρ y θ y la cantidad de rectas. Por esto, es razonable pensar en estos parámetros para realizar la clasificación de los estados. Debemos agrupar las rectas en *buckets*, a partir de la representación que nos devuelve la transformada de Hough y debemos definir cuántas rectas utilizaremos para representar el estado. Por ejemplo, para tener una cantidad de estados manejables podríamos tomar 5 buckets de ρ y 5 de θ y 3 rectas por imágen. Esto nos daría aproximadamente $\frac{(5\times5)^3}{3!} = \frac{15625}{6} = 2604$ estados matemáticamente posibles.

Como se puede observar, la clasificación de las situaciones en clusters puede ser muy útil. Sin embargo, puede suceder que la función de refuerzo provea refuerzos distintos para dos (o más) situaciones percibidas diferentes que están agrupadas en el mismo cluster [16]. Este problema recibe el nombre de *clustering alliasing*. Aprender es encontrar, en cada situación del entorno, la acción que tiene asociada la máxima suma de los refuerzos. Al agrupar las situaciones en clusters, podría pasar que no todas las situaciones agrupadas en un mismo cluster tengan el mismo refuerzo para la única acción asociada a ese cluster.

El problema de clustering alliasing dificulta el aprendizaje, y por lo tanto debemos intentar minimizarlo. Una opción para minimizar el clustering alliasing es modificar la función de refuerzo de manera de que en los clusters con alliasing sea posible encontrar una acción de compromiso. Otra opción es cambiar la manera en la que se hizo la clasificación, ya sea teniendo mayor granularidad para ρ o θ, mayor cantidad de rectas o agregando alguna otra información al estado, de manera de dividir los clusters problemáticos.

[1] Como vimos en la sección 3.2, $\theta \in [0, 180)$ y $\rho \in [-\frac{diag}{2}, \frac{diag}{2}]$. En nuestro caso $diag = \sqrt{640^2 + 480^2} = 800$.

En resúmen, nuestro método propone clasificar las situaciones para reducir la cardinalidad del espacio de estados agrupando las rectas en pocos *buckets* y utilizando un número pequeño de rectas para representar la situación percibida del entorno. Esto puede traer problemas de clustering alliasing que deben ser atendidos modificando la definición de la función de refuerzo y/o partiendo los clusters problemáticos. La granularidad de los parámetros ρ y θ, y cantidad de rectas que tomamos es una decisión fundamental que debe balancear la reducción del cardinal del espacio de estados con la minimización del clustering alliasing.

4.3.2. Cardinalidad del espacio de estados

Para poder elegir correctamente la granularidad de ρ, θ y cantidad de rectas es fundamental saber exactamente cuántos estados tendremos con esos valores. En general, una cota máxima para esta cantidad de estados viene dada por el cálculo combinatorio. Supongamos que tomamos n rectas con r valores diferentes de ρ y t valores diferentes de θ. Como pueden existir imágenes con menos de n rectas, podemos tener estados con $0, 1, ...n$ rectas. Entonces, debemos contar cuántos estados hay con 0 rectas, cuántos con 1 recta, cuántos con 2 rectas, y así hasta n. Finalmente tenemos que el cardinal del espacio de estados es:

$$\#S = \sum_{k=0}^{n} \#S_k,$$

donde S_k es el conjunto de los estados con k rectas. En las imágenes con cero rectas hay un único estado posible, entonces

$$\#S_0 = 1.$$

En las imágenes con una recta, hay $r \times t$ estados posibles, entonces

$$\#S_1 = r \times t.$$

En las imágenes con dos o más rectas tenemos que tener en cuenta que: 1) dos rectas que caigan en el mismo bucket son indistinguibles (o sea, no podemos tener dos rectas repetidas) y 2) el orden de las rectas no se tiene en cuenta. Luego,

$$\#S_2 = \frac{(r \times t)^2 - r \times t}{2!},$$

donde $(r \times t)^2$ es la suma de todas las posibles combinaciones de dos rectas; $-r \times t$ proviene de restarle todas las combinaciones con dos rectas repetidas y la división por 2! proviene de que el orden de las rectas no importa. En general, tenemos que:

$$\#S_n = \frac{(r \times t)^n - \sum_{k=0}^{n-1}(r \times t)^k}{n!}.$$

Por ejemplo, si tomamos como máximo 3 rectas por imágen y 5 valores diferentes de ρ y de θ, tendemos:

$$\begin{aligned} \#S &= \#S_0 + \#S_1 + \#S_2 + \#S_3 \\ &= 1 + \frac{(5 \times 5)^1}{1!} + \frac{(5 \times 5)^2 - \sum_{k=0}^{1}(5 \times 5)^k}{2!} + \frac{(5 \times 5)^3 - \sum_{k=0}^{2}(5 \times 5)^k}{3!} \\ &= 1 + 25 + 300 + 1975 = 2301. \end{aligned}$$

Esto nos da una cota máxima del tamaño del espacio de estados para cualquier tarea de AR que utilice este método, una vez fijado la cantidad de rectas, y la granularidad de ρ y θ.

En un entorno particular, no todas las combinaciones de rectas son factibles. Luego, dado un entorno particular podemos ajustar esta cota para tener una idea más precisa sobre la cantidad de estados reales que tendremos durante el aprendizaje.

Para poder tomar una decisión adecuada en cuanto al balance entre clustering alliasing y el cardinal del espacio de estados es necesario saber cuántos estados posibles realmente hay en el entorno. Para esto es útil tener una cota teórica máxima dada por la combinatoria de posibles rectas, pero se pueden lograr cotas mucho más ajustadas al realizar las pruebas de exploración durante el aprendizaje.

4.4. Percepción del entorno

En esta sección veremos en detalle los pasos a seguir para procesar las imágenes capturadas de forma de obtener las rectas que utilizaremos para construir la representación del entorno. A continuación presentamos el algoritmo utilizado a tal efecto:

```
img ← capturarImagen()
imgBorde ← sobel(img)
imgUmbral ← umbralConHisteresis(imgBorde,t0 ,t1)
H ← hough(imgBorde)
rectas ← maximos(H,cantRectas)
```

Figura 4.1: Algoritmo para obtener las rectas a partir de la imagen capturada.

El primer paso es capturar una imagen `img` en escala de grises. Luego aplicamos el operador de Sobel como se explica en la sección 3.3. La imagen de salida `imgBorde` es también una imagen en escala de grises que tiene el cálculo del gradiente para cada pixel.

Una vez que aplicamos el operador de Sobel, utilizamos umbralización con histéresis para determinar cuáles pixels son considerados parte de un borde. Como se explica en 3.3.2, la umbralización con histéresis es particularmente útil para la detección de bordes contínuos (como rectas) y eliminar los ruidos (como reflejos de luces en el piso o paredes). En este tipo de umbralización hay dos umbrales que fijar: t_0 y t_1. Estos umbrales deben ser valores entre 0 y 255, ya que la imagen a la que le aplicamos la umbralización es una imagen en escala de grises. La elección de estos umbrales depende fuertemente de las imágenes que queremos procesar. Fijar correctamente el valor de estos umbrales según el tipo de imágenes con el que estamos trabajando y al tipo de bordes que estamos buscando es fundamental para poder encontrar rectas en la imagen. La umbralización devuelve una imagen blanco y negro `imgUmbral` lista para que se pueda aplicar el algoritmo de Hough.

4.4.1. Hough

Como se explica en 3.2.1, lo primero que debemos hacer es discretizar los parámetros ρ y θ para poder implementar el algoritmo. En principio, y en función de tener una aproximación lo más cercana a las rectas reales, sería deseable contar con una granularidad lo más fina posible de ρ y θ. El problema es que una mayor granularidad implica un costo mayor de espacio en memoria y tiempo de procesamiento.

En cuanto al tiempo de procesamiento, como se explica en 3.2.1 el orden del algoritmo de Hough es $\mathcal{O}(n + mk)$ donde n es la cantidad de pixels de la imagen, $m \leq n$ es la cantidad de pixels en un borde y k es la cantidad de valores que puede tomar el ángulo θ. Como se puede ver, si la imagen es chica y/o tiene pocos bordes, entonces podemos tomar una granularidad más fina de θ.

En cuanto al espacio en memoria, como se explica en 3.2.1 tendremos un acumulador por cada celda (ρ_i, θ_j). Si llamamos t al cardinal de la partición

de θ y r al de ρ, podemos implementar esos acumuladores con una matriz H de dimensión $t \times r$.

Para definir la granularidad de θ y ρ deben tenerse en cuenta estas consideraciones en el caso particular al que se quiera aplicar el algortimo de Hough.

4.4.2. Detección de rectas

Una recta en la imagen original se traduce a un acumulador $H(\rho_i, \theta_j)$ con un valor alto, o sea, un máximo local de la matriz H. Como las imágenes no son un contínuo, sino que están divididas en pixels, y los parámetros ρ y θ están discretizados, una recta en la imagen produce toda una vecindad de acumuladores con un valor alto. Es por esto que una vez encontrado un máximo, debemos suprimir una vecindad de n filas y m columnas alrededor de este máximo, ya que esa vecindad corresponde a la misma recta.

El tamaño de la vecindad es crítico: si es demasiado chico, aparecerán rectas que no son parte de la imagen original y si es demasiado grande perderemos rectas que deberían aparecer. Al momento de fijar este parámetro debe tenerse en cuenta el tamaño de la imagen original y la discretización de ρ y de θ, ya que estos tres valores definen el tamaño de la matriz H.

Otro parámetro importante a fijar es el umbral u_r a partir del cual un acumulador $H(\rho_i, \theta_j)$ es considerado una recta. Además de este umbral absoluto es importante fijar otro relativo a la matriz H. Esto se puede hacer multiplicando el valor del acumulador mayor de H por un factor que llamaremos factor umbral c. Luego, consideraremos que un acumulador $H(\rho_i, \theta_j)$ pertenece a una recta sólo cuando:

$$H(\rho_i, \theta_j) > u_r \qquad \text{y} \qquad H(\rho_i, \theta_j) > c.max(H)$$

4.5. Técnica de exploración

En esta sección veremos cuál es la política a seguir para elegir las acciones a tomar durante el aprendizaje, es decir, la técnica de exploración. Como vimos en 2.9, las técnicas de exploración dirigida son muy útiles para disminuir el tiempo de exploración y, en el caso de tareas con un cardinal de estados elevado, pueden ser cruciales para la convergencia a una política satisfactoria. Siguiendo este enfoque, vamos a proponer una técnica de exploración dirigida libre de modelo.

4.5.1. Elección de acciones menos visitadas

En la sección 2.9 vimos que una técnica de exploración dirigida muy utilizada es el *counter-based*. La idea de esta técnica es visitar los estados menos recorridos. El problema es que requiere conocer la dinámica del entorno (la probabilidad $P^a_{ss'}$ de ir del estado s al estado s' al realizar la acción a).

Una posibilidad es aproximar $P^a_{ss'}$ dedicando una cantidad de iteraciones iniciales a explorar el entorno y construir la aproximación. El problema es que no queda claro que una cantidad de iteraciones al principio sean suficientes para tener una aproximación aceptable de $P^a_{ss'}$.

Una alternativa al counter-based que no requiere conocer $P^a_{ss'}$ puede ser en lugar de elegir una acción que nos lleve a un estado menos visitado, elegir la acción menos ejecutada. Este dato es fácil de mantener: para cada par estado acción (s, a) recordamos cuántas veces ejecutamos la acción a en el estado s hasta el momento.

Esta idea combinada con la técnica ϵ-greedy (ver sección 2.9) resulta en la siguiente política:

```
elegir mejor acción con probabilidad 1 − ϵ.
elegir acción menos ejecutada con probabilidad ϵ.
```

Figura 4.2: Primera versión para la política utilizada durante el aprendizaje.

4.5.2. Exploración tardía

Un problema que encontramos es que a veces durante el aprendizaje se visita por primera vez un estado tardíamente, es decir, cuando el ϵ de la técnica ϵ-greedy es bajo. Esto produce que la primera acción que se elije en ese estado queda marcada como mejor acción y no se exploren el resto de las acciones para ese estado. La próxima vez que se visita ese estado, el ϵ es aún menor y por lo tanto la probabilidad ϵ de elejir una acción poco visitada es muy pequeña. Para solucionar este problema pensamos en tener un valor de ϵ por estado. Para cada estado s_i tenemos un ϵ_i que se decrementa sólo cuando es visitado el estado s_i.

Combinando esta técnica con las ideas anteriores, tenemos la siguiente política para la elección de la acción a tomar en el estado s_i:

Se toma el máximo entre el ϵ_{gral} y el ϵ_i para privilegiar la exploración. Por un lado, si un estado es visitado por primera vez cuando el aprendizaje ya está avanzado, el ϵ_{gral} es bajo pero el ϵ_i alto, por lo que se explora ese estado como si hubiera sido visitado al principio del aprendizaje.

```
ϵ ← max(ϵi , ϵgral ).
elegir mejor acción con probabilidad 1 − ϵ.
elegir acción menos ejecutada con probabilidad ϵ.
```

Figura 4.3: Segunda versión para la política utilizada durante el aprendizaje.

Por otro lado, la distribución de los estados no tiene porqué ser uniforme en el ambiente. Por lo tanto, los estados que suceden en el ambiente con una mayor frecuencia son más visitados durante el aprendizaje. La mejor manera de implementar un ϵ_i por estado es, entonces, conocer la frecuencia de cada estado s_i y ajustar la velocidad de disminución del ϵ_i del estado acorde a esa frecuencia. Si un estado s_i es muy frecuente, entonces es más visitado durante el aprendizaje y por lo tanto el ϵ_i debe tener una velocidad de disminución más lenta. Sin embargo, esto implica conocer parte de la dinámica del entorno: debemos conocer la frecuencia en la que ocurren los estados. Por lo tanto, usaremos la misma velocidad de disminución para todos los ϵ_i. De esta manera el ϵ_i para los estados muy frecuentes disminuye demasiado rápido, pero al tomar el máximo entre el ϵ_{gral} y el ϵ_i aseguramos que la exploracion sea satisfactoria igualmente.

En cada caso particular se debe ajustar la velocidad con que es disminuidos el ϵ_i y el ϵ_{gral}. Como ϵ_i sólo tiene en cuenta las veces que fue visitado el estado s_i, debe decrementarse más rápidamente que el ϵ_{gral}.

4.6. Función de refuerzo

Como vimos en 2.10, la función de refuerzo es la manera de especificar los objetivos de una tarea en AR. Por lo tanto, depende fuertemente de la tarea que debe aprenderse. Es por eso que, más allá de los lineamientos generales presentados en la sección 2.10, la definición de esta función deberá hacerse en cada caso particular al que se desee aplicar el método.

Capítulo 5

Experiencias en simulación

El este capítulo se muestran las experiencias realizadas en simulación, utilizando el método propuesto en el capítulo anterior, para un caso particular de estudio. Primero presentaremos los objetivos perseguidos con estas experiencias. Luego, describiremos los métodos y materiales utilizados, analizando en detalle cómo se aplica el método propuesto en el caso de estudio. Por último, presentaremos los resultados obtenidos.

5.1. Objetivo

Para el presente trabajo se desarrolló un entorno de simulación con dos objetivos. Por un lado, a partir de las experiencias en simulación podemos validar la hipótesis de trabajo y el método propuesto. Queremos ver que efectivamente es posible utilizar la transformada de Hough para reducir el cardinal de estados de las imágenes extrayendo la información necesaria para utilizar la técnica de AR Q-Learning.

Por otro lado, para validar nuestra hipótesis y el método propuesto con un robot real, puede ser muy provechoso contar con un entorno de simulación. Podemos ajustar los parámetros de nuestro método (cantidad de rectas, cantidad de buckets de ρ y θ, etc), así como los de Q-Learning (valores de α, γ) y los de la técnica de exploración (ϵ y velocidad de decaimiento), antes de realizar las experiencias con el robot real. Uno de los objetivos que perseguimos con las experiencias en simulación es encontrar una configuración de estos parámetros que permita minimizar la cantidad de iteraciones necesarias para la etapa de aprendizaje. Esto es fundamental para poder realizar las experiencias con un robot real, ya que el costo de realizar experiencias con un robot real es muy alto (cada iteración lleva más tiempo que en simulación, la batería que alimenta al robot tiene un tiempo de carga de varias horas, etc). La simulación nos provee un entorno de trabajo mucho más sencillo de manejar que un sistema real. Se

pueden realizar pruebas en varias computadoras a la vez, es más rápido y no requiere que un operario esté presente durante la experiencia.

Una condición necesaria para utilizar simulación es mantener una representación que se aproxime lo más posible a la realidad. Es decir, que respete la semántica de las situaciones que se presentan en el sistema real. En nuestro caso, utilizamos un modelo cinemático, que se ajusta adecuadamente a la realidad, para definir las interacciones del robot con el entorno. Es esta condición la que nos permite utilizar un modelo simple para la simulación respetando las condiciones del sistema real.

5.2. Primera experiencia: recorrer un pasillo simple

5.2.1. Motivación

El robot real con el que vamos a realizar las experiencias es un robot móvil. Por eso, la tarea a realizar va a incluir acciones que involucren movimientos. Además, necesitamos trabajar con un entorno que permita capturar los aspectos principales con las rectas inferidas a partir de las imágenes capturadas con la cámara. Por lo anterior, elegimos como primera experiencia recorrer un pasillo simple con un robot movil.

5.2.2. Materiales y métodos

Descripción del simulador

El desarrollo del simulador, así como el de la aplicación que implementa el algoritmo de Q-Learning se realizó en el lenguaje de programación `Java`. Por su orientación a objetos `Java` resulta muy útil a la hora de definir un entorno y un robot simulados, así como también los distintos elementos del algoritmo de Q-Learning (los estados, la matriz para almacenar los valores de Q, la función de refuerzo, etc). La otra característica que se tomó en cuenta para elegir `Java` como lenguaje de desarrollo es la independencia de plataforma. Esto significa que los programas escritos en `Java` pueden ejecutarse igualmente en cualquier tipo de hardware o de sistema operativo, sin necesidad de que sean recompilados. A diferencia de los lenguajes de programación convencionales, que generalmente están diseñados para ser compilados a código nativo, un programa escrito en `Java` es compilado en un *bytecode* que es interpretado por una máquina virtual. Esta característica de portabilidad fue fundamental en nuestro trabajo, ya que permitió correr varias pruebas de simulación en varias máquinas diferentes (con distintas arquitecturas y distintos sistemas

operativos) sin tener que modificar o recompilar el código. De la misma forma utilizamos el mismo código para el algortimo de Q–Learning en el robot real, sin tener que realizar ninguna modificación.

Para el desarrollo del simulador se utilizó la librería gráfica `Java3d` que utiliza `OpenGL`. `Java3D` nos permite escribir programas que despliegan gráficas tridimensionales e interactuar con ellas. Contiene un conjunto de constructores de alto nivel para crear y manipular geometrías en 3D, así como de técnicas que permiten dibujar la geometría en una imagen. Contiene funciones necesarias para crear imágenes, visualizaciones, animaciones y aplicaciones con objetos gráficos interactivos. Decidimos utilizar esta librería porque nos permite crear un ambiente 3D estático, situar una cámara y moverla en el ambiente con directivas simples.

El simulador diseñado no es de propósito general. Está pensado para construir pasillos, ya que nuestras experiencias son en este tipo de ambientes. Para eso el simulador tienen una clase `Pared` que nos permite construir paredes de distintos tamaños, texturas, colores. Luego, tiene una clase `Pasillo` que se genera a partir de un conjunto de paredes y que también contiene a la cámara. El simulador tiene una clase `CapturingCanvas3D` que nos permite capturar la imagen de 313×213 pixels a partir de la posición en la que se encuentra la cámara.

El simulador es manejado con un driver implementado en la clase `SimAdapter` que provee las directivas de mover, girar y sensar el entorno. Por último, el robot es simulado con la clase `Robot` que utiliza el `SimAdapter` [1] para sensar el entorno y ejecutar la acción que corresponda según se esté en la etapa de aprendizaje o de explotación de la política.

Descripción de la librería de procesamiento de imágenes

Para realizar el procesamiento de las imágenes desarrollamos una librería en `ISO C++`. La elección de este lenguaje de programación estuvo basada en que, por un lado necesitamos trabajar a bajo nivel para procesar las imágenes con rapidez, y por el otro, necesitamos trabajar a alto nivel para encapsular la funcionalidad de cada paso del procesamiento de imágenes. Si bien esto último también puede hacerse en `ANSI C`, la utilización de clases resulta una herramienta muy útil para el encapsulamiento. Por eso, decidimos que `C++` era el lenguaje más apropiado para esta parte del trabajo.

Esta librería implementa la función de Sobel, la función de umbralización con histéresis y la transformada de Hough. Estas funciones se combinan como se explica en la figura 4.1 para obtener las rectas que utilizamos en la

[1] En el caso de las experiencias con el robot real se utiliza la clase `RealAdapter` que tiene implementada la misma interfaz que la clase `SimAdapter` y que se comunica con la librería de control del robot real.

representación del entorno.

5.2.3. El ambiente y la tarea

En esta primera experiencia, el ambiente es un pasillo de 4 metros de longitud y 1 metro de ancho. Las paredes del pasillo tienen 1 metro de alto. La pared al final del pasillo tiene una puerta centrada de 50 cm de alto y 40 cm de ancho. Incluímos esta puerta para garantizar la presencia de rectas en las imágenes, a lo largo de todo el pasillo. La cámara se sitúa a 35 cm de altura, para corresponderse con la cámara que posee el robot real. En la figura 5.1 podemos ver un diagrama del pasillo.

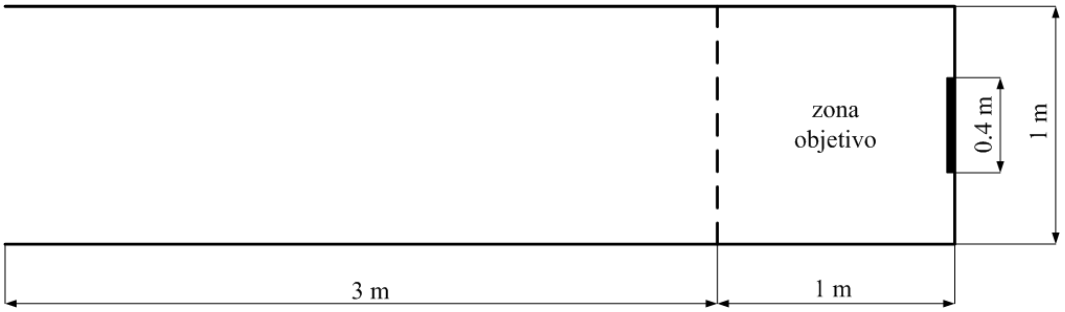


Figura 5.1: Diagrama del pasillo simple utilizado para la primera experiencia.

La tarea a realizar en esta primera experiencia consiste en llegar desde cualquier posición inicial válida en el pasillo hasta una posición que supere los tres metros (zona objetivo), sin chocar con las paredes. Una posición inicial es válida cuando el estado percibido en dicha posición es válido (ver 5.2.5).

5.2.4. Las acciones

A continuación mostramos las acciones que puede realizar el agente en cada instante:

Acción	**Descripción**
Avanzar	avanza 10 cm en línea recta
Retroceder	retrocede 10 cm en línea recta
Girar derecha	gira 10 grados en sentido horario
Girar izquierda	10 grados en sentido antihorario

Cuadro 5.1: Conjunto de acciones que puede realizar el agente.

Los giros sólo modifican el ángulo de visión del agente, sin que éste avance ni retroceda. Los valores de avance y giro se fijaron experimentalmente. Dentro

de las experiencias preliminares fueron considerados giros de 5 y 15 grados y avances de 7 y 13 cm. En función del criterio de efectividad definido en la sección 5.2.7 para las políticas obtenidas, elegimos los valores mostrados anteriormente.

5.2.5. Representación del entorno

Ajuste de parámetros para el procesamiento de las imágenes

En la sección 4.4 describimos el procedimiento seguido para obtener las rectas a partir de las imágenes capturadas del entorno, y describimos los parámetros que deben ser ajustados. A continuación mostramos los valores de estos parámetros utilizados para las experiencias en simulación. Estos valores fueron encontrados de manera experimental, siguiendo lo explicado en la sección 4.4.

Etapa	**Parámetro**	**Valor**
Detección de bordes (umbralización)	t_0	150
	t_1	200
Transformada de Hough	ρ	uno por pixel
	θ	uno por grado
Detección de rectas (vecindad y umbralización)	n	$\#filas(H)/10$
	m	$\#cols(H)/10$
	u_r	20
	c	0,25

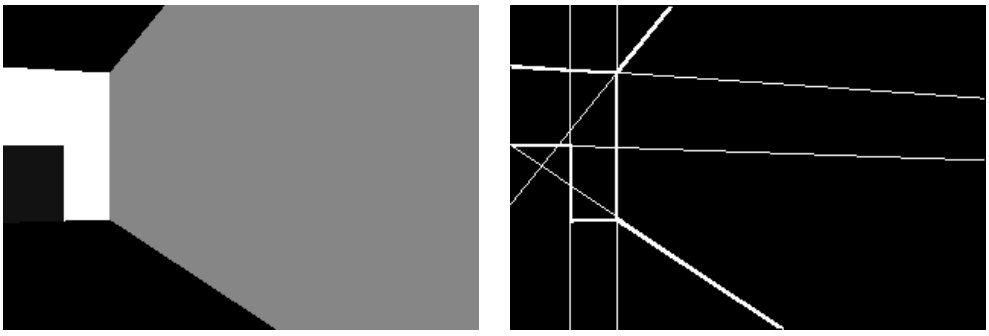
Cuadro 5.2: Parámetros del procedimiento seguido para obtener las rectas a partir de las imágenes, en simulación.

Tomar una granularidad de un ρ por pixel y un θ por grado en una imagen de 313×213 pixels nos da 180 valores de θ diferentes y $diag = \lceil\sqrt{313^2 + 213^2}\rceil = 379$ valores de ρ (ver 3.2), definiendo una matriz de acumuladores H de 180 columnas y 379 filas (ver 4.4.1). Como vimos en la sección 4.4.2, la definición del tamaño de la vecindad depende de las dimensiones de la matriz H. Experimentalmente se notó que una vecindad que fuera de un décimo de la cantidad de filas y columnas de la matriz H es lo suficientemente grande como para que no aparezcan rectas que no son parte de la imagen original y lo suficientemente chica como para no perder rectas que sí deben aparecer. Por lo tanto, en las experiencias en simulación se adoptó: $n = \lceil 379/10 \rceil = 38$ y $m = \lceil 180/10 \rceil = 18$. Además, para la detección de rectas fijamos el umbral absoluto u_r en 20 y el factor c para el umbral relativo en 0,25.

A continuación mostramos tres imágenes capturadas por la cámara en el pasillo simple simulado y sus correspondientes imágenes procesadas según los parámetros anteriores.

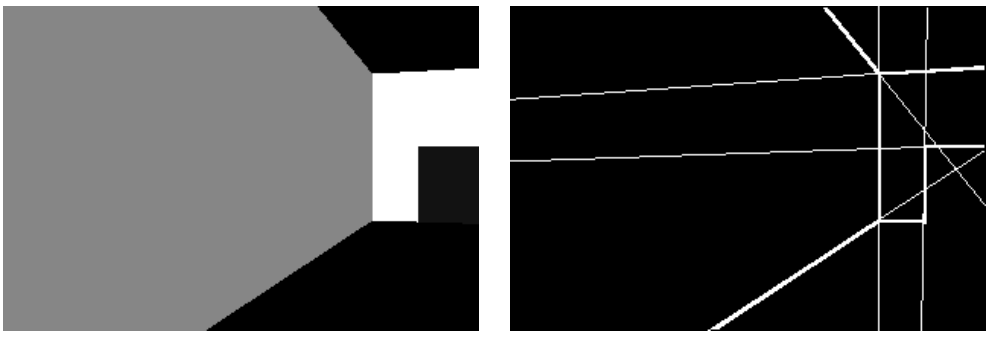
a) b)

Figura 5.2: a) Imagen capturada desde el inicio del pasillo, con 0 grados de orientación. b) Imagen procesada.



a) b)

Figura 5.3: a) Imagen capturada desde el inicio del pasillo, con 20 grados a la derecha de orientación. b) Imagen procesada.



a) b)

Figura 5.4: a) Imagen capturada desde el inicio del pasillo, con 20 grados a la izquierda de orientación. b) Imagen procesada.

Espacio de estados

Para nuestra representación del entorno, un estado es un conjunto de rectas. Cada recta va a estar representada por una dupla (ρ, θ) que denota el bucket

correspondiente a ρ y θ. Si tenemos k_ρ buckets de ρ y k_θ buckets θ entonces un estado s va estar definido como:

$$s = \Big\{(\rho_0, \theta_0), (\rho_1, \theta_1), (\rho_2, \theta_2), \ldots, (\rho_{n-1}, \theta_{n-1})\Big\}$$

donde n es la cantidad de rectas que tomamos por imagen, $0 \leq \rho_i < k_\rho$ y $0 \leq \theta_i < k_\theta$ con $0 \leq i < n$.

Si en la imagen existen dos rectas que caen en el mismo bucket ρ_i y θ_i, la representación del estado sólo contiene una tupla (ρ_i, θ_i). Es decir, las rectas no se repiten en el estado. Además, vamos a definir como estados válidos aquellos donde la cantidad de rectas sea $n > 1$. Tomamos esta definición ya que un estado con una o cero rectas no tiene suficiente información del entorno como para poder asociar una acción de manera razonable.

Es importante aclarar que la representación del entorno adoptada requiere que tratemos al entorno como no determinístico, ya que define funciones de transición $P^a_{ss'}$ no determinísticas. Por ejemplo, la acción Avanzar a veces deja al agente en el mismo estado (porque las rectas obtenidas a partir de las imágenes sensadas caen en los mismos buckets que en la posición anterior) y a veces lleva al agente a otro estado.

En la sección 5.2.7 de este mismo capítulo analizaremos resultados con distintos valores para la cantidad de rectas, buckets de ρ y buckets de θ.

Información adicional a los estados

Además del conjunto de rectas, necesitamos alguna información adicional. Esta información no forma parte de los estados (y por ende no se almacena como parte de la función Q), pero es necesaria para la función de refuerzo que vamos a utilizar durante el aprendizaje. Por un lado, necesitamos conocer la posición (o una aproximación) del robot en cada instante para saber si ya llegamos al objetivo. Por otro lado, necesitamos saber si el robot choca contra alguna pared luego de realizar una acción. La primera información la podemos obtener directamente del simulador, que nos dice a cada instante la posición exacta del robot simulado. La segunda información también se puede obtener directamente del simulador. Pero en este caso, se optó por simular sensores de distancia que se activan cuando el robot está suficientemente cerca de una pared. Esto es así para que la interfaz del robot simulado sea la misma que la que tiene el robot real. Para más detalle a cerca de cuál es la interfaz del robot real ver la sección 6.1.1.

Situaciones inválidas

Mientras el robot aprende puede llegar a situaciones inválidas. Por un lado, esto ocurre cuando el robot se acerca lo suficiente a una pared para considerar que chocó. Por otro lado, puede llegar a situaciones con estados inválidos asociados (estados con una o cero rectas). En estos casos se activa un *reflejo* para que el robot realice la acción contraria a la que lo llevó a la situación inválida. Esto es posible ya que por las características del entorno las acciones definidas son reversibles.

5.2.6. La función de refuerzo

El objetivo de la tarea definida es recorrer el pasillo hasta llegar a una región llamada zona objetivo (ver figura 5.1). De esto se desprende una primera función de refuerzo:

$$r(a, s, s') = \begin{cases} k \in \Re_{>0} & \text{si el robot llega al objetivo} \\ 0 & \text{sino.} \end{cases} \tag{5.1}$$

Es importante aclarar que cuando nos referimos a un estado s en el contexto de la función de refuerzo lo hacemos agregándole la información adicional (la posición instantánea del robot y el estado de activación de los sensores de distancia).

Sin embargo, como se vio en la sección 2.10, si trabajamos sólo con refuerzos puramente demorados, y esos refuerzos no ocurren con mucha frecuencia, se corre el riesgo de que el algoritmo de Q-Learning no funcione correctamente. Fue por este motivo que decidimos incorporar estimadores de progreso a la función de refuerzo para introducir refuerzos inmediatos. Este recurso ya fue explotado por otros autores tales como Mataric en [17]. Estos estimadores de progreso nos indican si al ejecutar una acción nos acercamos, nos alejamos o nos mantenemos equidistantes del objetivo. Luego, la función de refuerzo resulta:

$$r(a, s, s') = \begin{cases} k \in \Re_{>0} & \text{si el robot llega al objetivo} \\ p \in \Re_{>0} & \text{si disminuye la distancia al objetivo} \\ -p & \text{si aumenta la distancia al objetivo} \\ 0 & \text{sino.} \end{cases} \tag{5.2}$$

Por último tenemos que definir la función de refuerzo para los casos en los que se activa un reflejo, producto de que el robot llegó a una situación inválida.

Como queremos que el robot aprenda a evitar estas situaciones inválidas, asignamos un valor negativo para los casos en los que se activa un reflejo. Luego, la función de refuerzo completa queda de la siguiente forma:

$$r(a,s,s') = \begin{cases} r \in \Re_{<0} & \text{si se activa un reflejo} \\ k \in \Re_{>0} & \text{si el robot llega al objetivo} \\ p \in \Re_{>0} & \text{si disminuye la distancia al objetivo} \\ -p & \text{si aumenta la distancia al objetivo} \\ 0 & \text{sino.} \end{cases} \tag{5.3}$$

Los valores de k, p y r se fijaron experimentalmente en 50, 1 y -1 respectivamente. También se realizaron experiencias con valores de $k = 100$, $r = -10$ y $p = 0{,}1$. Si bien es importante mantener una amplia diferencia entre el valor del refuerzo demorado k y el de los refuerzos inmediatos (estimadores de progreso y reflejos), pudimos notar en las experiencias preliminares que las variaciones en estos valores no afectan el desempeño de la función de refuerzo.

5.2.7. Resultados

En esta sección se presentan los resultados obtenidos al aplicar el método propuesto con diferentes valores para: la cantidad de rectas, la cantidad de buckets de ρ y la cantidad de buckets de θ.

Consideraciones previas

Lo primero que vamos a definir para realizar las experiencias son los valores de los parámetros del algoritmo de Q-Learning. Como trabajamos en un entorno no determinístico los valores de la función Q pueden variar mucho durante el aprendizaje. Por eso vamos a trabajar con un α chico, que nos garantiza mayor estabilidad, aunque menor rapidez en el aprendizaje. Por otro lado, como cada vez que ejecutamos una acción, nos interesa tener en cuenta tanto el refuerzo inmediato como a futuro, fijamos un γ que contemple esto. Por eso, y en función de las experiencias previas que realizamos para determinar estos valores, fijamos α en 0,1 y γ en 0,5.

Lo segundo que tenemos que definir es un criterio para comparar las políticas obtenidas con distintas cantidades de rectas, de buckets de ρ y de θ. Es decir, tenemos que fijar un criterio de *efectividad* de las políticas aprendidas con las ditintas configuraciones de parámetros. Para eso, vamos a tener en cuenta dos cosas al testear una política: 1) el porcentaje de veces que el agente llega al objetivo y 2) el porcentaje de estados visitados durante el test que no fueron recorridos durante el aprendizaje. Para esto último, tenemos que

determinar la cantidad de estados que resultan una vez elegida una configuración de parámetros. Como se vio en la sección 4.3.2 la cantidad de estados matemáticamente posibles puede ser bastante superior a la cantidad de estados reales. Por eso, vamos a utilizar los estados recorridos durante la etapa de aprendizaje como una aproximación a la cantidad de estados reales del entorno. Si bien existen estados en el entorno que no son recorridos durante el aprendizaje, lo que nos interesa analizar es cuántos de los estados visitados durante el testeo de la política no fueron recorridos durante el aprendizaje. En este sentido, resulta útil tomar esta aproximación de la cantidad de estados reales del entorno.

En las pruebas que realizamos variamos la cantidad de rectas, la cantidad de buckets de ρ y de θ. En la siguiente tabla podemos ver en detalle cada configuración de parámetros:

Configuración de parámetros	**Cantidad de rectas**	**Cantidad buckets de ρ**	**Cantidad buckets de θ**
p1	3	5	5
p2	3	10	5
p3	3	5	10
p4	4	5	5
p5	4	10	5
p6	4	10	10

Cuadro 5.3: Configuraciones de parámetros para las experiencias realizadas en simulación.

En los gráficos que se muestran en este capítulo las curvas para cada configuración de parámetros muestran políticas representativas de una serie de experiencias.

A continuación mostramos un histograma con la cantidad de estados recorridos durante el aprendizaje para cada configuración de parámetros.

Para poder testear una política debemos situar al robot en una posición inicial y ejecutar desde allí la política. Si el robot llega al objetivo consideramos que es un episodio exitoso. Si el robot llega a una situación inválida (choca contra una pared [2] o llega a un estado con menos de dos rectas) o entra en un ciclo de acciones que lo dejan en el mismo estado, consideramos que es un episodio fallido.

Puede ser que el agente llegue a un estado que no fue recorrido durante el aprendizaje, es decir, un estado para el cual no existe una acción asociada en la política. En este caso se ejecuta la acción Adelante (acción por defecto) y

[2]Mientras testeamos una política, fijamos la distancia de activación de los sensores de distancia en 0, o sea, cuando realmente se choca contra una pared.

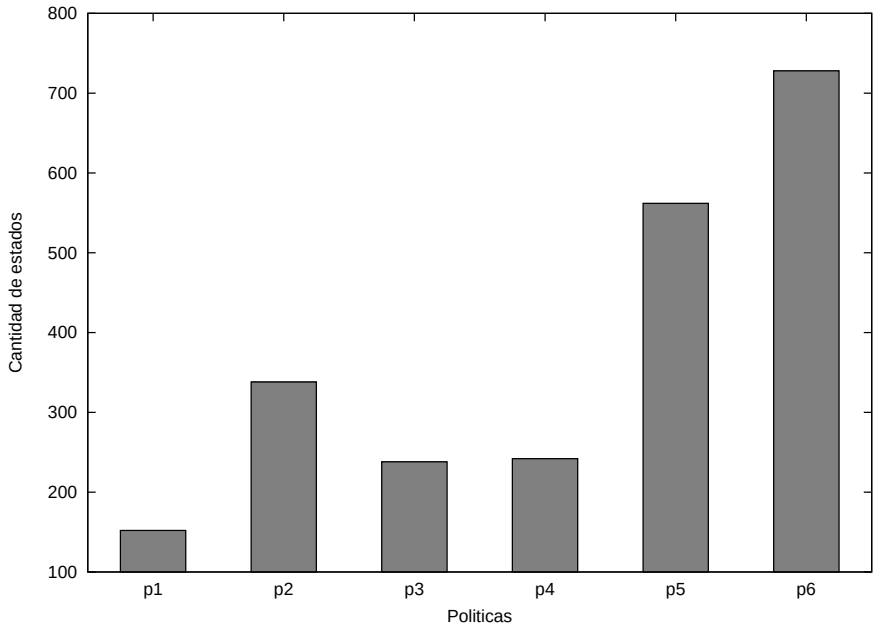


Figura 5.5: Histograma de la cantidad de estados recorridos durante el aprendizaje para cada configuración de parámetros.

se marca ese estado como no visitado.

Por último, tenemos que definir cuáles y cuántas son las posiciones iniciales para comenzar los episodios durante el test. Cuando nos referimos a una posición inicial estamos diciendo que tenemos que fijar una posición (x, y) en el pasillo con un ángulo de orientación del robot.

Decidimos tomar un conjunto de posiciones iniciales aleatorias que sean una muestra del espacio de posiciones del entorno. Para definir qué cantidad de posiciones iniciales es muestra, tomamos la configuración de parámentros que resulta en la mayor cantidad de estados (4 rectas, 10 buckets de ρ y 10 de θ) y testeamos con 300 posiciones iniciales aleatorias dos políticas aprendidas con esa configuración de parámetros. Cada 10 posiciones iniciales guardamos los resultados para ver dónde se estabiliza el porcentaje de episodios exitosos. En la figura 5.6 podemos ver estos resultados.

Como vemos a partir de 180 posiciones inciales, el porcentaje de episodios exitosos se estabiliza, y esto guarda independencia con respecto a la cantidad de iteraciones utilizadas para obtener la política durante el aprendizaje. Por todo lo anterior, vamos a tomar como muestra un conjunto de 180 posiciones iniciales al azar, para realizar los tests de las diferentes políticas aprendidas.

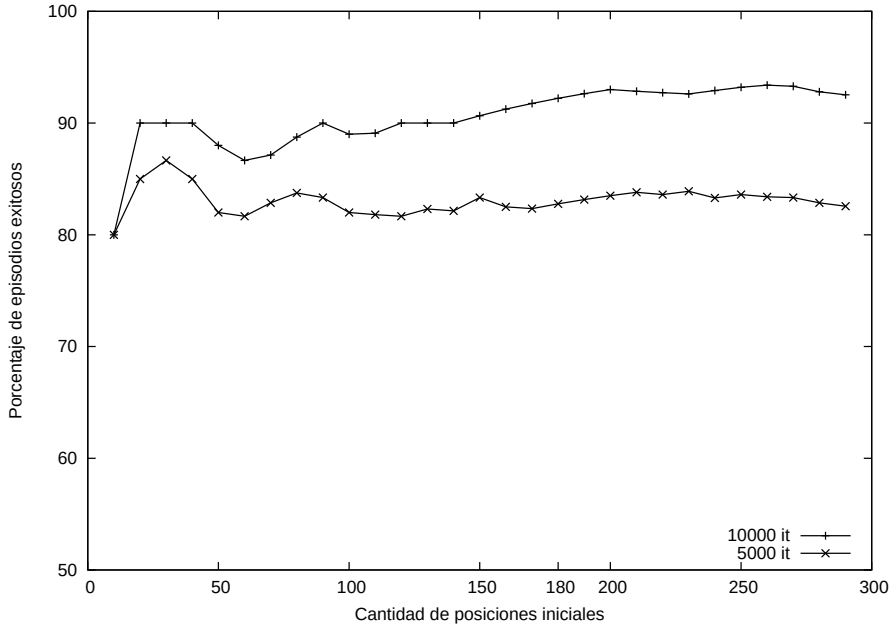


Figura 5.6: Porcentaje de episodios exitosos, para distintas cantidades de posiciones iniciales aleatorias con dos políticas obtenidas con 5.000 y 10.000 iteraciones durante el aprendizaje.

Comparación de configuraciones de parámetros

Una vez que fijamos una muestra, podemos comparar la efectividad de las políticas aprendidas con las distintas configuraciones de parámetros, tanto en porcentaje de veces que llega al objetivo como en porcentaje de estados que recorre durante el test que no fueron visitados durante el aprendizaje.

El gráfico de la figura 5.7 muestra el porcentaje de episodios exitosos de las políticas aprendidas con las diferentes configuraciones en función de la cantidad de iteraciones. El decaimiento del ϵ general se fijó para que llegue a $0,05$ (es decir $0,5\,\%$) en 10.000 iteraciones y el decaimiento del ϵ_i por estado se fijó para que llegue a $0,05$ en 30 visitas al estado s_i.

En la figura 5.7 podemos observar que todas las configuraciones llevan a políticas satisfactorias en 10.000 iteraciones. Sin embargo, con una cantidad menor de iteraciones, las mejores políticas obtenidas son aquellas cuya configuraciones de parámetros tienen asociadas una menor cantidad de estados. Además, podemos observar que para estas configuraciones el porcentaje de episodios exitosos se estabiliza a partir de las 6.000 iteraciones.

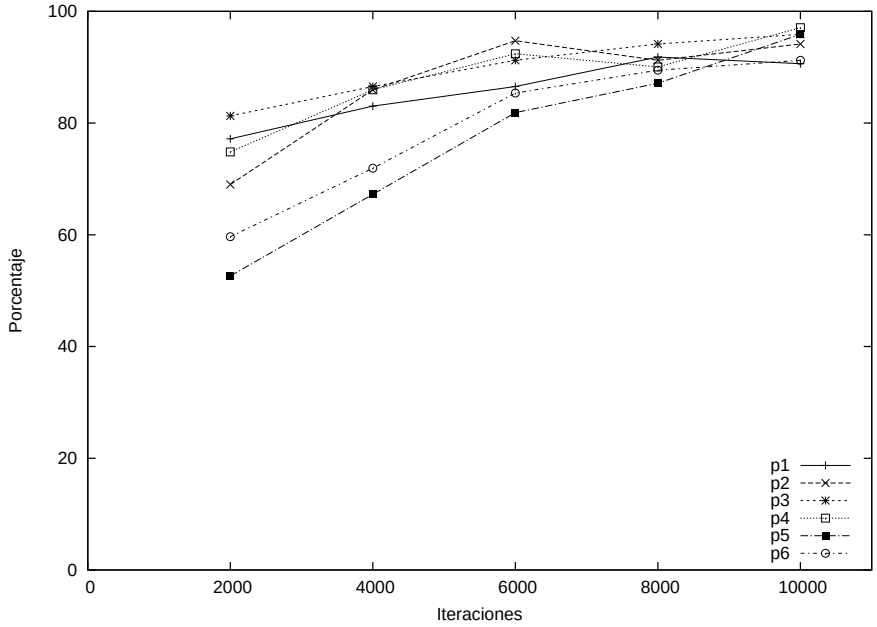


Figura 5.7: Porcentaje de episodios exitosos en función de la cantidad de iteraciones. Comparación entre las distintas configuraciones de parámetros.

En la figura 5.8 se puede ver la comparación de las configuraciones con respecto al porcentaje de estados visitados que no fueron recorridos durante el aprendizaje.

Como podemos observar en la figura 5.8, todas las políticas obtenidas con 10.000 iteraciones tienen un porcentaje de estados no recorridos aceptable (menor al 10 %). Nuevamente las políticas obtenidas con configuraciones que definen una cardinalidad menor del espacio de estados son mejores. La configuración **p1** (3 rectas, 5 buckets de ρ y 5 buckets de θ) es la que define las políticas con menor porcentaje de estados no recorridos.

Como dijimos en la sección 5.1, un objetivo que perseguimos con las experiencias en simulación es fijar los parámetros del método propuesto para minimizar la cantidad de iteraciones necesarias para aprender una política aceptable. Queremos que las experiencias en simulación sirvan como un paso previo a las experiencias con un robot real, donde es fundamental contar con una cantidad de iteraciones durante el aprendizaje que hagan plausible de realizar la experiencia.

Por eso, una vez analizados los datos anteriores, realizamos nuevas experiencias de aprendizaje con menor cantidad de iteraciones. Decidimos utilizar

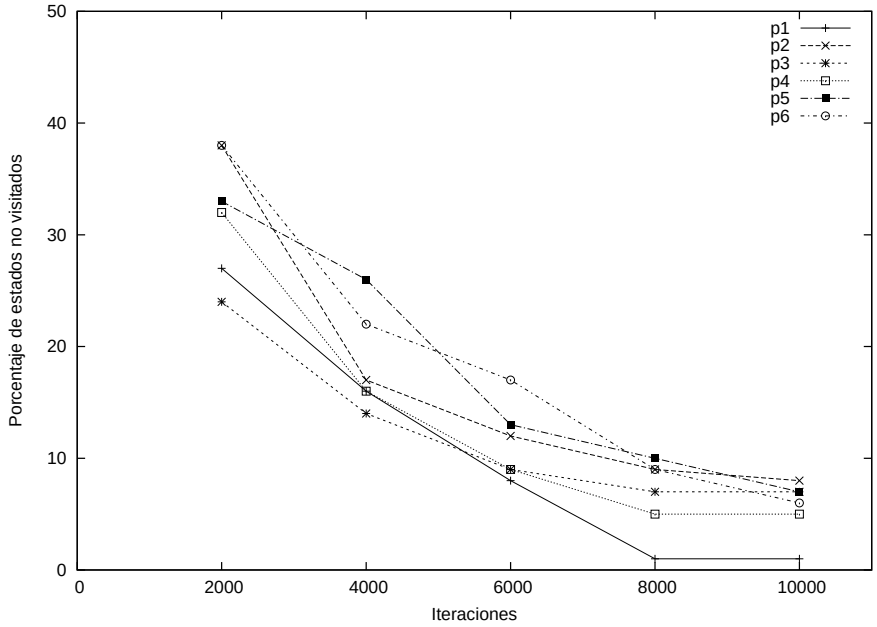


Figura 5.8: Porcentaje de estados que no fueron recorridos durante el aprendizaje en función de la cantidad de iteraciones. Comparación entre las distintas configuraciones de parámetros.

6.000 iteraciones ya que en este valor notamos que el porcentaje de episodios exitosos se estabiliza en las políticas aprendidas.

Para estas experiencias utilizamos las configuraciones de 3 rectas: **p1**, **p2** y **p3**. Elejimos testear la configuración **p1** (3 rectas, 5 buckets de ρ y 5 buckets de θ) porque es la que tiene asociada la menor cantidad de estados y el menor porcentaje de estados no visitados. Elejimos testear la configuración **p2** (3 rectas, 10 buckets de ρ y 5 buckets de θ) porque tiene la política con mejor porcentaje de episodios exitosos en 6.000 iteraciones y resulta también en una cantidad pequeña de estados. Finalmente, decidimos testear la configuración **p3** (3 rectas, 5 buckets de ρ y 10 buckets de θ) porque es la que tiene la curva más suave en el porcentaje de episodios exitosos, con máximos en 2000 y 8000 iteraciones.

Utilizamos un ϵ general que decae a $0,05$ en 6.000 iterciones. El ϵ por estado sigue siendo igual que en las experienias anteriores. Los resultados se muestran a continuación en las figuras 5.9 y 5.10:

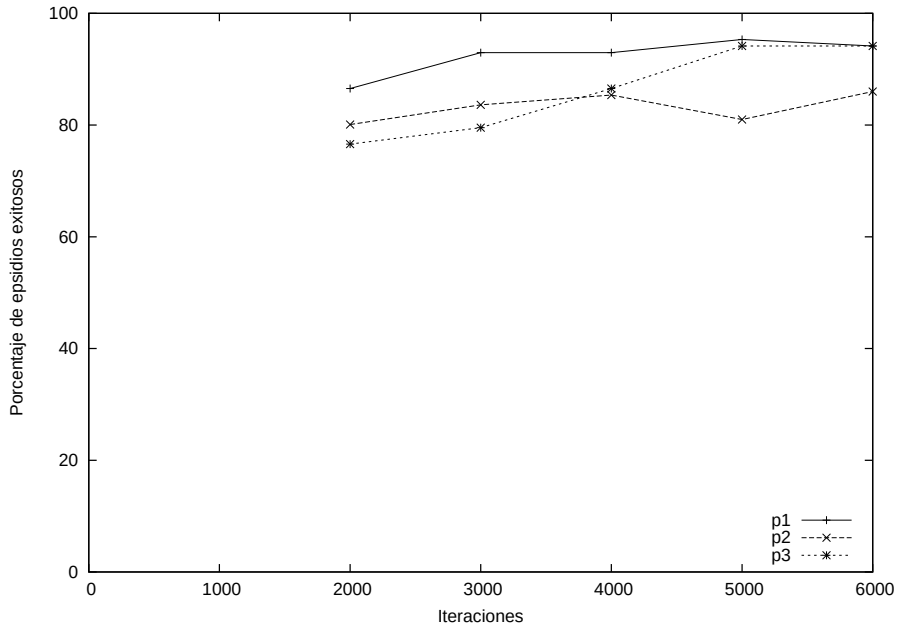


Figura 5.9: Porcentaje de episodios exitosos en función de la cantidad de iteraciones. Comparación entre las distintas configuraciones de parámetros.

Como vemos la configuración con 3 rectas, 5 buckets de ρ y 5 de θ da lugar a las políticas más eficientes en términos del criterio de comparación adoptado.

Por último, podemos comparar los resultados de aplicar la técnica de exploración $\epsilon - greedy$ y la desarrollada para el método propuesto. Para eso realizamos la experiencia de aprender políticas con una configuración de 3 rectas, 5 buckets de ρ y 5 de θ utilizando la técnica de exploración $\epsilon - greedy$ (que llamaremos **p7**) y la comparamos con las políticas obtenidas utilizando la técnica de exploración propuesta en el método (**p1**). Estas pruebas se realizaron con un ϵ general que decae a $0,05$ en 6.000 iteraciones.

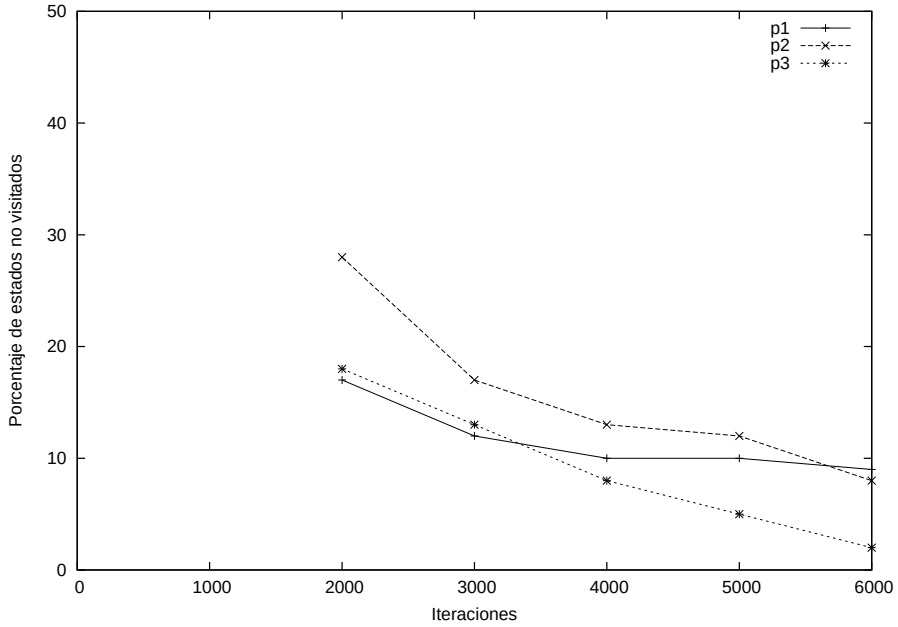


Figura 5.10: Porcentaje de estados que no fueron recorridos durante el aprendizaje en función de la cantidad de iteraciones. Comparación entre las distintas configuraciones de parámetros.

Como podemos observar en la figura 5.11 la técnica de exploración desarrollada en el método propuesto disminuye la cantidad de iteraciones necesarias para aprender una política satisfactoria.

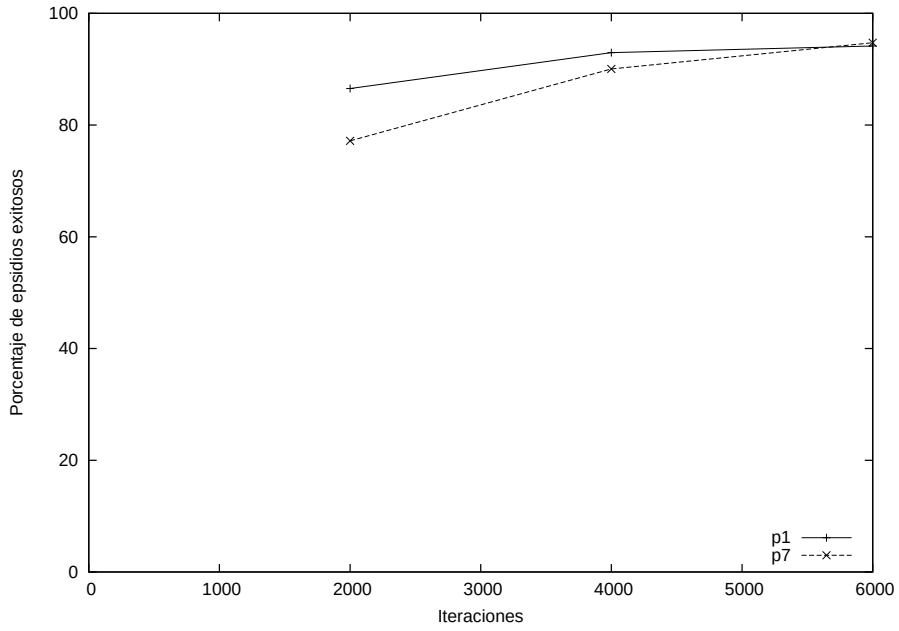


Figura 5.11: Porcentaje de episodios exitosos en función de cantidad de iteraciones de aprendizaje. Comparación entre la técnica de exploración $\epsilon - greedy$ y la desarrollada para el método propuesto para la configuración de 3 rectas, 5 buckets de ρ y 5 de θ.

5.3. Segunda experiencia: recorrer un pasillo cruzado

5.3.1. Motivación

En la sección anterior mostramos que el método propuesto funciona para un caso simple. Sin embargo, en ese caso se pueden usar otros sensores para llevar a cabo la tarea. Por ejemplo, se pueden utilizar sensores infrarrojos de proximidad para aprender a seguir paredes (*wall following*) a una determinada distancia. De esta forma, se puede cumplir con el objetivo de recorrer el pasillo, siguiendo una pared a una determinada distancia hasta la zona objetivo.

Una de las motivaciones para abordar el AR con percepción basada en visión es que existen entornos y tareas donde el sensor más adecuado es una cámara de video. La motivación de la siguiente experiencia es utilizar el método propuesto en un caso de estudio donde otros sensores (como los sensores de proximidad) no resultan suficientes.

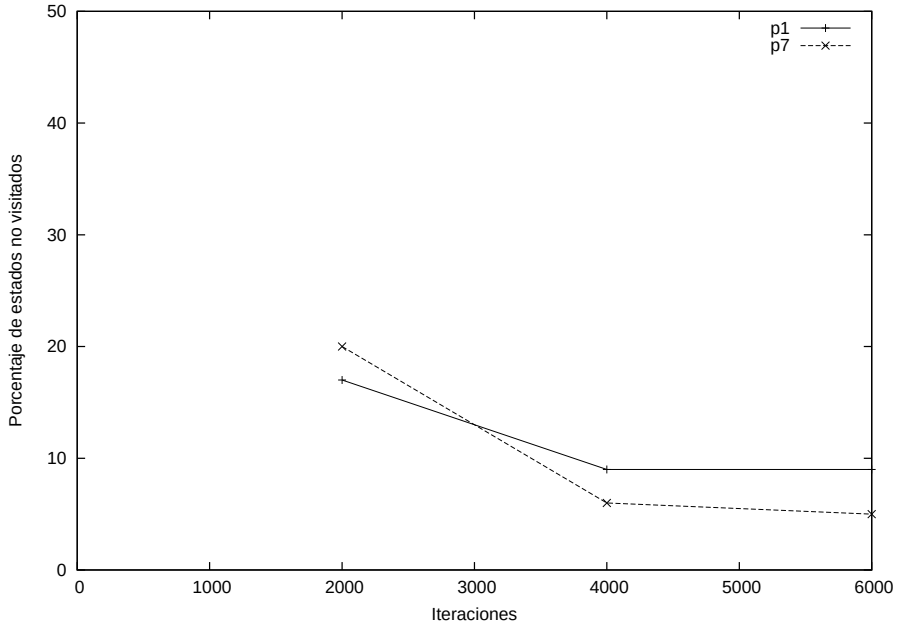


Figura 5.12: Porcentaje de estados que no fueron recorridos durante el aprendizaje. Comparación entre la técnica de exploración $\epsilon - greedy$ y la desarrollada para el método propuesto para la configuración de 3 rectas, 5 buckets de ρ y 5 de θ.

5.3.2. Materiales y métodos

Para esta segunda experiencia se utilizó el mismo simulador, la misma librería para el procesamiento de las imágenes (con los mismos parámetros). También se utilizaron las mismas acciones definidas para el agente, la misma función de refuerzo y la misma representación para los estados.

5.3.3. El ambiente y la tarea

Para esta segunda experiencia, incluimos otro pasillo que cruza al pasillo original de forma perpendicular. Con este nuevo ambiente, un robot dotado únicamente de sensores de proximidad que aprende a seguir una pared tomaría el pasillo lateral, y no alcanzaría el objetivo.

Definimos un pasillo de 4 metros de longitud, 1 metro de ancho y 1 metro de altura. Este pasillo está cruzado por otro en forma perpendicular entre el primer y segundo metro. El pasillo perpendicular también tiene un metro de ancho y paredes de 1 metro de altura. La pared al final del pasillo mantiene la

puerta centrada de 50 cm de alto y 40 cm de ancho. La cámara se sitúa a 35 cm de altura. En la figura 5.13 podemos ver un diagrama del pasillo utilizado para la segunda experiencia.

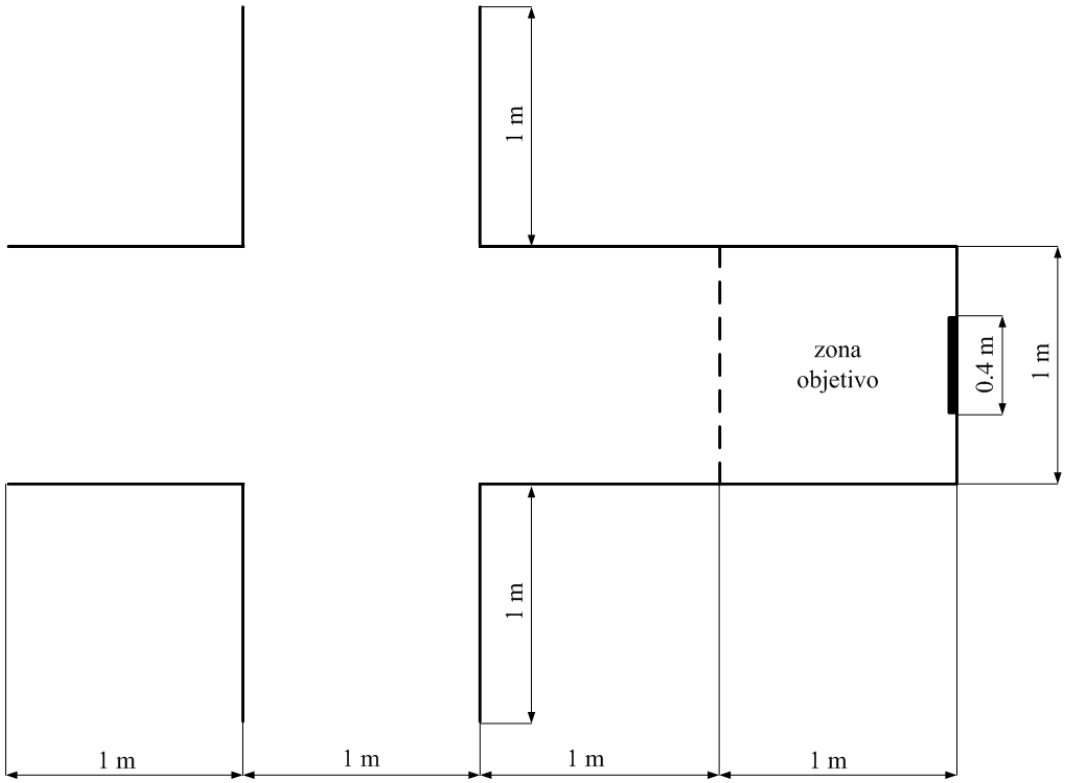
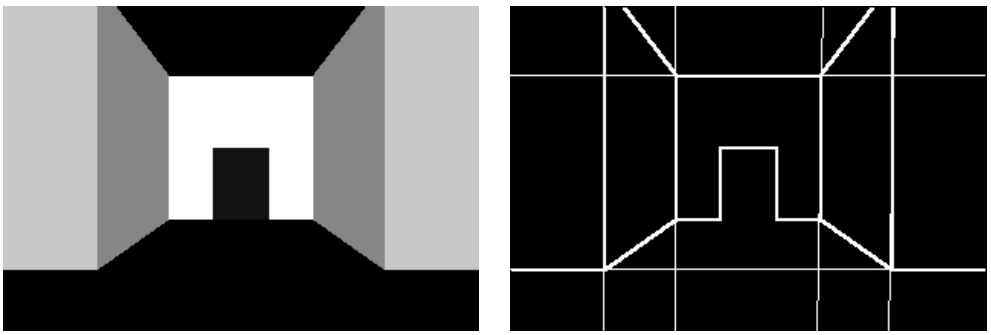


Figura 5.13: Diagrama del pasillo cruzado utilizado para la segunda experiencia.

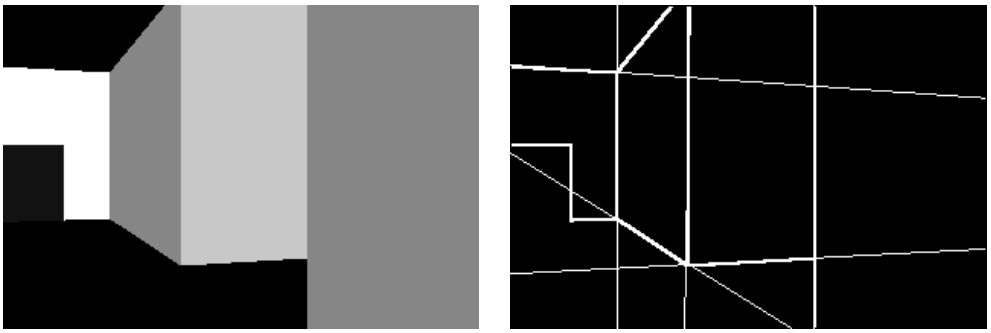
La tarea a realizar sigue siendo llegar desde cualquier posición inicial válida en el pasillo hasta una posición que supere los tres metros (zona objetivo), sin chocar con las paredes.

En las figuras 5.14, 5.15 y 5.16 mostramos tres imágenes capturadas por la cámara en este pasillo simulado y sus correspondientes imágenes procesadas con los parámetros definidos en la tabla 5.2.



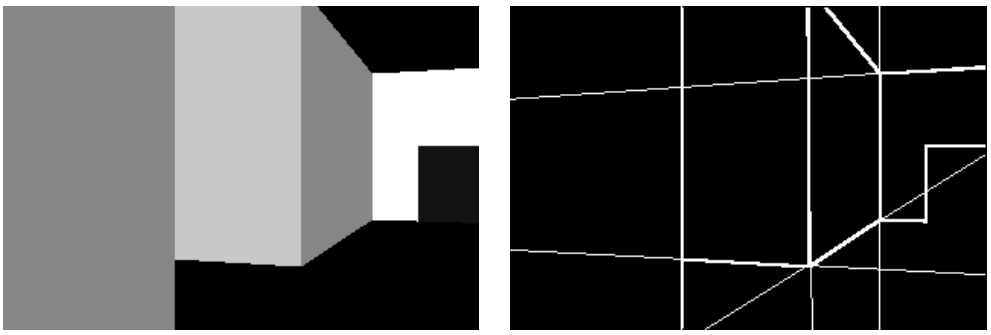
a) b)

Figura 5.14: a) Imagen capturada desde el inicio del pasillo, con 0 grados de orientación. b) Imagen procesada.



a) b)

Figura 5.15: a) Imagen capturada desde el inicio del pasillo, con 20 grados a la derecha de orientación. b) Imagen procesada.



a) b)

Figura 5.16: a) Imagen capturada desde el inicio del pasillo, con 20 grados a la izquierda de orientación. b) Imagen procesada.

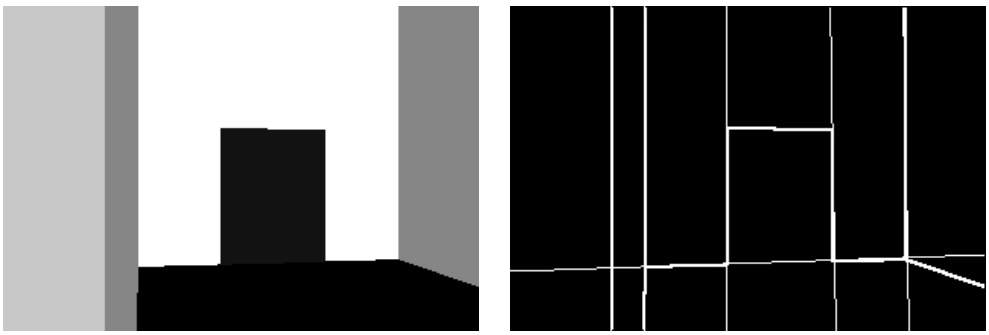
5.3.4. Resultados

Consideraciones previas

Para elegir los parámetros del método propuesto para esta segunda experiencia, partimos de los resultados obtenidos en la primera experiencia. Como vimos en la sección anterior, las configuraciones con 3 rectas resultaron muy eficientes de acuerdo al criterio adoptado para realizar los tests. Como el ambiente del segundo caso de estudio es más complejo, vamos a tener mayor cardinalidad del espacio de estados. Necesitamos elegir entonces, una configuración de los parámetros del método propuesto que posea una mayor granularidad para distinguir situaciones diferentes. Por lo anterior, una experiencia preliminar que realizamos utiliza la configuración de parámetros **p2** (3 rectas, 10 buckets de ρ y 5 de θ) que es la configuración de 3 rectas que tiene asociada la mayor cantidad de estados en el caso del pasillo simple.

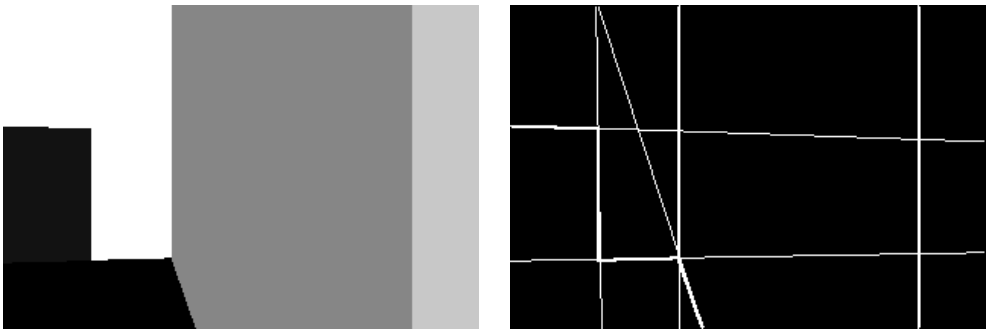
Al aumentar la complejidad del ambiente, necesitamos una mayor cantidad de iteraciones para el aprendizaje. Por lo tanto, tomamos 32.000 iteraciones para las experiencias preliminares. Para testear la política utilizamos 220 posiciones iniciales, ya que esta cantidad resulta ser una muestra de las posiciones iniciales (ver figura 5.21). Los resultados de testear la política obtenida con 3 rectas, 10 buckets de ρ, 5 de θ y 32.000 iteraciones fueron: porcentaje de episodios exitosos 63,60 % y porcentaje de estados no recorridos durante el aprendizaje $3,56\,\%$.

A partir de estos resultados, analizamos las causas del bajo porcentaje de episodios exitosos que obtuvimos con esta política. Siguiendo el camino trazado en las experiencias, observamos que con esos parámetros hay problemas de cluster alliasing que impiden llegar a una política aceptable. Por ejemplo, las dos situaciones de las figuras 5.17 y 5.18 están asociadas al mismo estado.



a) b)

Figura 5.17: a) Imagen capturada sobre la izquierda del pasillo principal a la altura del pasillo lateral con orientación de 10 grados hacia la derecha. b) Imagen procesada.



a) b)

Figura 5.18: a) Imagen capturada sobre la derecha del pasillo principal a la altura del pasillo lateral, con orientación de 10 grados hacia la derecha. b) Imagen procesada.

En la posición de la figura 5.17, la acción que debería elegir el agente es girar a la derecha, ya que girar a la izquierda implica adentrarse en el pasillo lateral, e ir hacia adelante implica chocar contra la pared del pasillo lateral. Por otro lado, la acción de retroceder tiene refuerzo negativo ya que el agente se aleja del objetivo.

En la posición de la figura 5.18, la acción que debería elegir el agente es girar a la izquierda, ya que girar a la derecha implica adentrarse en el pasillo lateral, e ir hacia adelante implica ir hacia la pared del pasillo lateral. Por otro lado, la acción de retroceder tiene refuerzo negativo ya que el agente se aleja del objetivo.

Luego, estamos ante dos situaciones con refuerzos diferentes que están representadas por un mismo estado. En estas situaciones, las mejores acciones a tomar son contrapuestas (girar a la derecha y a la izquierda) y no podemos

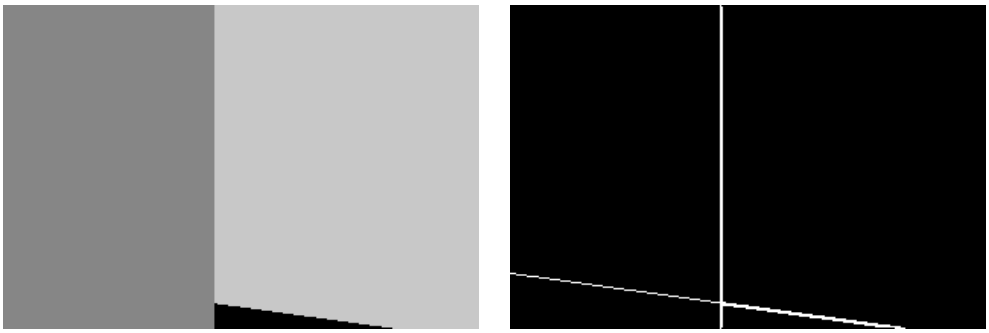
encontrar una acción de compromiso, ya que las otras dos acciones (ir hacia adelante o hacia atrás) son malas.

Otras dos situaciones representadas con un mismo estado son las situaciones de las figuras 5.19 y 5.20.

a) b)

Figura 5.19: a) Imagen capturada sobre la derecha del pasillo principal a la altura del pasillo lateral, con orientación de 30 grados hacia la derecha. b) Imagen procesada.

En la posición de la figura 5.19, la acción que debería elegir el agente es girar a la izquierda, ya que girar a la derecha o avanzar implican dejar de percibir la recta inferior e ir a una situación con un estado inválido asociado (una imagen con sólo una recta). Por otro lado, la acción de retroceder tiene refuerzo negativo ya que el agente se aleja del objetivo. La única acción con refuerzo positivo en esta situación es girar a la izquierda.



a) b)

Figura 5.20: a) Imagen capturada sobre la izquierda del pasillo principal a la altura del pasillo lateral, con orientación de 30 grados hacia la izquierda. b) Imagen procesada.

En la posición de la figura 5.20, la acción que debe elegir el agente es girar a la derecha, ya que girar a la izquierda o avanzar implican dejar de

percibir la recta inferior e ir a una situación con un estado inválido asociado (una imagen con sólo una recta). Por otro lado, la acción de retroceder tiene refuerzo negativo ya que el agente se aleja del objetivo. La única acción con refuerzo positivo en esta situación es girar a la derecha.

Nuevamente, estamos ante dos situaciones con refuerzos diferentes que están representadas por un mismo estado. Las mejores acciones a tomar son contrapuestas (girar a la derecha y a la izquierda) y no podemos encontrar una acción de compromiso, ya que el resto de las acciones, en cada caso, tienen refuerzos negativos.

En el caso de las figuras 5.17 y 5.18, podemos partir este cluster tomando una recta más como parte del estado. Sin embargo, esto no soluciona el problema que se presenta en las situaciones 5.19 y 5.20. En estas situaciones tomar como parte del estado una recta más no cambia el estado, ya que las imágenes originales poseen sólo dos rectas. Decidimos entonces incluir en los estados la orientación en la que es capturada la imagen, discretizada en 8 buckets de 45 grados cada uno.

Ambas decisiones (tomar una recta más e incluir la orientación) son necesarias. Asi como tomar una recta más no soluciona el problema que se presenta en las situaciones 5.19 y 5.20, tener en cuenta la orientación no soluciona el problema que se presenta en las situaciones 5.17 y 5.18 ya que ambas imágenes son tomadas con el mismo ángulo.

En función de todo lo anterior, en las pruebas que realizamos para el pasillo cruzado, tomamos la configuración de parámetros de 4 rectas, 10 buckets de ρ, 5 buckets de θ y 8 buckets para la orientación de la imagen.

Para poder comparar las políticas en este nuevo entorno, debemos definir nuevamente un conjunto de posiciones iniciales que sea muestra. Para definir qué cantidad de posiciones iniciales es muestra, tomamos la configuración de parámetros que resulta en la mayor cantidad de estados (4 rectas, 10 buckets de ρ y 5 de θ y 8 buckets de orientación de la imagen) y testeamos con 400 posiciones iniciales aleatorias la política aprendida en 32.000 iteraciones. Cada 10 posiciones iniciales guardamos los resultados para ver dónde se estabiliza el porcentaje de episodios exitosos. En la figura 5.21 podemos ver estos resultados.

Como vemos, a partir de 220 posiciones inciales, el porcentaje de episodios exitosos se estabiliza. Por todo lo anterior, vamos a tomar como muestra para esta segunda experiencia, un conjunto de 220 posiciones iniciales aleatorias, para realizar los tests de las diferentes políticas aprendidas.

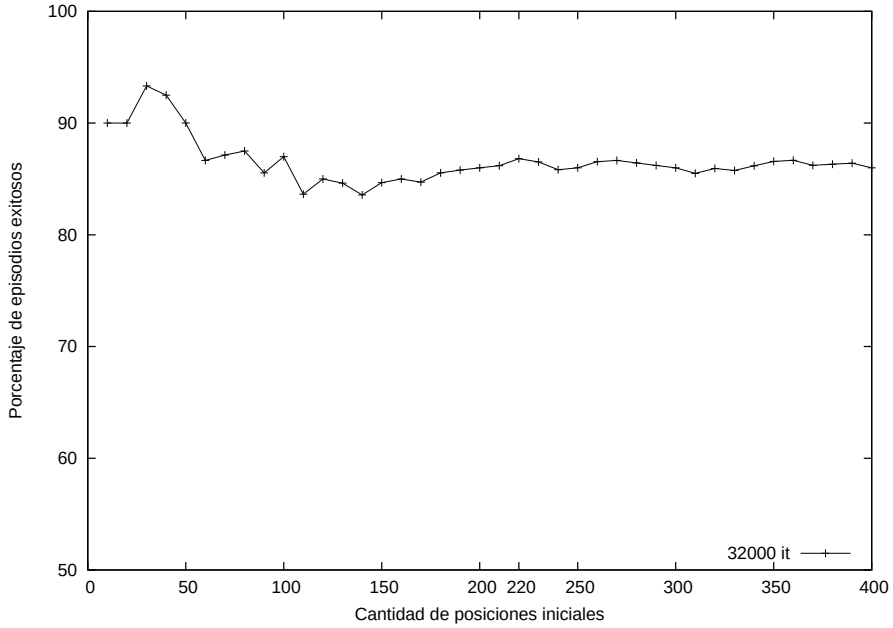


Figura 5.21: Porcentaje de episodios exitosos, para distintas cantidades de posiciones iniciales aleatorias utilizando la política aprendida con 4 rectas, 10 buckets de ρ, 5 buckets de θ, 8 buckets de orientación y 32.000 iteraciones.

Comparación de las políticas

A continuación vamos a comparar los resultados de aplicar la técnica de exploración $\epsilon - greedy$ y la desarrollada para el método propuesto. Como dijimos anteriormente vamos a trabajar con 4 rectas, 10 buckets de ρ, 5 de θ y 8 buckets de orientación para las imágenes. En los gráficos, la curva **p1** muestra las políticas aprendidas utilizando la técnica de exploración desarrollada para nuestro método y **p2** las políticas obtenidas utilizando la técnica de exploración $\epsilon - greedy$.

En primer lugar, podemos observar que a 32.000 iteraciones en ambas experiencias se logra un política satisfactoria con más del 87 % de episodios exitosos y menos del 5 % de estados recorridos durante el test que no fueron visitados durante el aprendizaje.

También podemos observar en la figura 5.22 que la técnica de exploración desarrollada mejora la velocidad de aprendizaje. En 10.000 iteraciones la política aprendida con esta técnica tiene el $73,27\,\%$ de episodios exitosos mientras que la política aprendida utilizando la técnica $\epsilon - greedy$ tiene sólo el $53,91\,\%$. Finalmente, podemos observar en la figura 5.23 que la técnica de

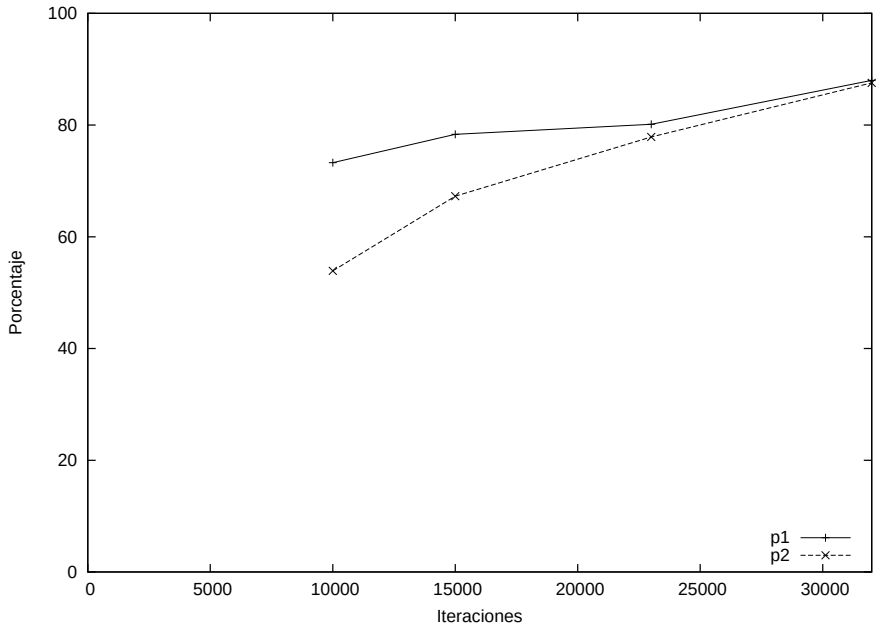


Figura 5.22: Porcentaje de episodios exitosos en función de cantidad de iteraciones de aprendizaje. Comparación entre la técnica de exploración $\epsilon - greedy$ y la desarrollada para el método propuesto para la configuración de 4 rectas, 10 buckets de ρ, 5 buckets de θ y 8 buckets de orientación para las imágenes.

exploración desarrollada mejora también la cantidad de estados explorados. En un mismo ambiente y con igual cantidad de rectas, buckets de ρ y de θ, utilizando nuestra técnica se recorrieron durante el aprendizaje 1,663 estados mientras que con la técnica $\epsilon - greedy$ se recorrieron 1,450 estados.

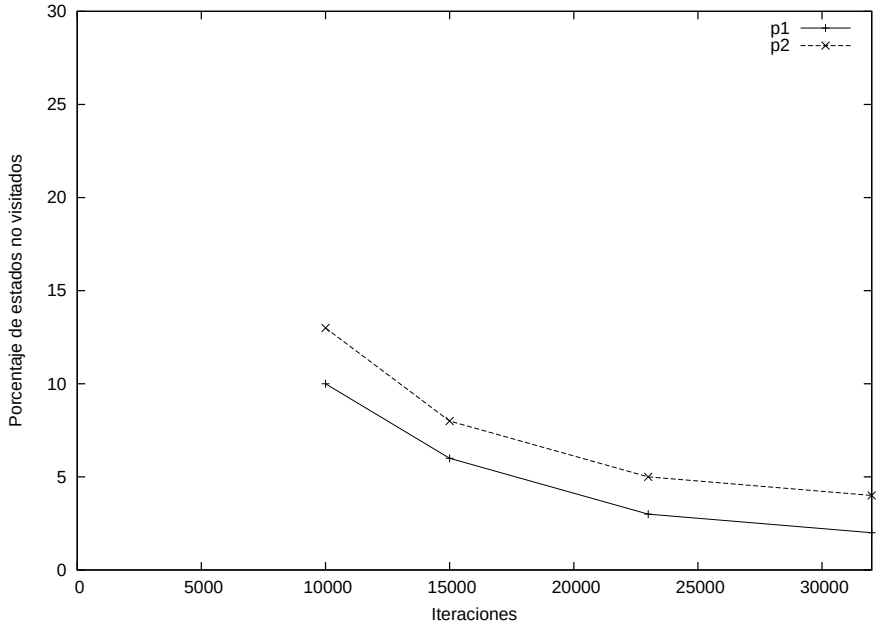


Figura 5.23: Porcentaje de estados que no fueron recorridos durante el aprendizaje. Comparación entre la técnica de exploración $\epsilon - greedy$ y la desarrollada para el método propuesto para la configuración de 4 rectas, 10 buckets de ρ y θ y 8 buckets de orientación para las imágenes.

5.4. Conclusiones de las experiencias en simulación

Con las experiencias en simulación pudimos validar el método propuesto para un caso de estudio simple, que involucra un agente móvil en un ambiente sencillo. Luego, pusimos a prueba el método propuesto en un ambiente donde una percepción del entorno basada en otros sensores (como por ejemplo, sensores de proximidad) no es suficiente. En ambos casos, los resultados fueron muy satisfactorios. Se comprobó que un número pequeño de rectas (3 o 4) es suficiente para obtener una representación del ambiente apropiada para utilizar el algortimo de Q-Learning, por lo menos para los casos estudiados.

La técnica de exploración dirigida propuesta no requiere conocer la dinámica del entorno, es decir, es libre de modelo. Esta característica hace que sea apropiada para utilizar en la etapa de aprendizaje del método propuesto. En las experiencias en simulación, pudimos observar que utilización de esta técnica disminuye considerablemente la cantidad de iteraciones necesarias para aprender una política satisfactoria. Esta técnica puede ser aplicada en el desarrollo

de otros comportamientos de robots autónomos que utilicen AR.

A partir de un análisis más detallado de las experiencias podemos extraer las siguientes conclusiones. La cantidad de estados asociados a las distintas configuraciones de parámetros no crece exponencialmente al aumentar la cantidad de rectas, buckets de ρ o de θ como sugería la fórmula matemática para la cantidad de estados (ver 4.3.2).

En el pasillo simple pudimos observar que todas las configuraciones presentadas dan lugar a políticas satisfactorias para 10.000 iteraciones. Las configuraciones con menor cantidad de estados asociados resultan en políticas aceptables en menor cantidad de iteraciones. Las políticas aprendidas a partir de estas configuraciones se estabilizan a partir de las 6.000 iteraciones respecto del test de episodios exitosos.

Como buscamos minimizar la cantidad de iteraciones requeridas para el aprendizaje, realizamos experiencias con las configuraciones de 3 rectas disminuyendo el ϵ general para que llegue a 0,05 en 6.000 iteraciones. En este caso, observamos que la configuración de 3 rectas, 5 buckets de ρ y 5 de θ da lugar a políticas más eficientes en términos de episodios exitosos. Las políticas aprendidas a partir de estas configuraciones se estabilizan a partir de las 3.000 iteraciones.

En la última experiencia que realizamos con esta configuración en el pasillo simple, comparamos la técnica de exploración *ϵ-greedy* y la desarrollada para el método propuesto. Observamos que la técnica de exploración desarrollada disminuye la cantidad de iteraciones necesarias para aprender una política satisfactoria.

La primera experiencia en el pasillo cruzado la realizamos utilizando la configuración de 3 rectas, 10 buckets de ρ y 5 de θ. Observamos que esta configuración no da lugar a políticas satisfactorias dado que se presentan situaciones de clustering alliasing. Para resolver este problema fue necesario tomar una configuración de 4 rectas y además incorporar una nueva información al estado: la orientación de las imágenes sensadas. Este último dato se discretizó en 8 buckets de 45 grados cada uno. Las pruebas realizadas con esta nueva configuración dieron políticas satisfactorias en 32.000 iteraciones de aprendizaje.

En la última experiencia realizada en el pasillo cruzado comparamos la técnica de exploración $\epsilon - greedy$ y la desarrollada para el método propuesto con la configuración de 4 rectas, 10 buckets de ρ, 5 de θ y 8 para la orientación. Observamos que 32.000 iteraciones en ambas experiencias se logra un política satisfactoria. También observamos que la técnica de exploración disminuye la cantidad de iteraciones requeridas durante el aprendizaje para obtener una política satisfactoria. Finalmente, para el caso del pasillo cruzado, la técnica de exploración desarrollada aumenta la cantidad de estados explorados, tanto

en porcentaje de estados no visitados, como en el total de estados recorridos durante el aprendizaje.

Capítulo 6

Experiencias con el robot FenBot

En este capítulo describiremos las experiencias realizadas y los resultados obtenidos con el robot real FenBot para la tarea de seguir un pasillo simple.

6.1. Materiales y métodos

6.1.1. El robot Fenbot

El robot FenBot fue desarrollado por el Laboratorio de Robótica del Departamento de Computación de la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires, a cargo del grupo de investigación ICAR (Inteligencia Computacional Aplicada a la Robótica). El robot FenBot tiene un cuerpo con forma de octaedro, de 10 cm de lado y 23 cm de altura (ver figura 6.1). Es un robot autónomo, con varias unidades de procesamiento, de sensado y actuación.

Unidad de procesamiento y almacenamiento

El FenBot cuenta con una PC embebida con un procesador Intel Celeron Pentium III de 700 Megahertz y 256 Megabytes de memoria SDRAM. Tiene una interface PCI, una interface IDE, dos puertos USB, dos puertos de comunicación Ethernet, dos puerto series (uno RS232 y otro UART TTL), un puerto paralelo y uno infrarrojo. Tiene una controladora de gráficos VGA/SXGA con salida de video. La placa mother se alimenta con 5 y 3,3 volts. Además, tiene un disco rígido Ultra DMA de 30 Gigabytes de capacidad que se comunica al mother por la interface IDE. La PC tiene instalado un sistema operativo `GNU Linux` con una versión de kernel 2.4.27-2-386.

Visión

El FenBot cuenta con dos cámaras digitales color del fabricante THE IMAGING SOURCE modelo DFK 21F04. Son cámaras FireWire (estandar IEEE 1394) que poseen un chip CCD de 1/4 de pulgada de escaneo progresivo. Tienen una resolución de 640×480 pixeles y pueden transmitir hasta 30 imágenes por segundo. Se conectan a la placa mother del robot mediante una placa FireWire siguiendo el protocolo de transmisión de datos DCAM (sin compresión de datos ni canal de audio). Estas cámaras poseen lentes intercambiables lo que posibilita elegir la más apriapiada para cada aplicación. Se alimenatan con 12 volts. Para las experiencias realizadas en este trabajo se utilizó sólo una de las dos cámaras.

Sensores de distancia

El FenBot tiene un anillo de 16 telémetros infrarrojos en el perímetro externo del robot. En cada lateral se ubican dos sensores: uno de corto y otro de largo alcance. La figura 6.1 muestra un esquema de la distribución de los sensores. Estos sensores son utilizados para medir distancias entre el robot y objetos que se encuentren en el entorno. Tienen un led que emite un haz de luz infarrojo y un receptor que detecta el reflejo. A partir del ángulo de incidencia que tiene el reflejo en el receptor, se calcula la distancia al objeto. La sensibilidad puede variar considerablemente para distintos sensores, y puede verse afectada por condiciones externas, como por ejemplo por una fuente de luz que afecte al receptor [28].

Los sensores de corto alcance son modelo GP2D120 del fabricante de componentes electrónicos SHARP y permiten sensar distancias entre 4 y 40 cm. Los de largo alcance son modelo GP2Y0A02YK del mismo fabricante que permiten sensar distancias entre 20 y 150 cm. La salida de estos sensores es analógica, por lo que el FenBot tiene una placa que controla estos sensores y traduce la señan analógica de cada sensor en una señal digital. Esta placa se comunica con el mother del robot mediante el puerto de comunicación serie. Cada sensor tiene una resolución de 1 byte, es decir, que devuelven valores entre 0 y 255. Una característica de estos sensores que se debe tener en cuenta es que el valor que devuelven no mantiene necesaramiente una relación lineal con la distancia entre el sensor y el objeto. En fabricante proporciona una curva que determina para cada distancia (en cm) el valor que entrega cada tipo de sensor.

Actuadores

El FenBot utiliza para moverse dos ruedas independientes que se encuentran ubicadas en el perímetro externo del robot, como puede verse en la figura

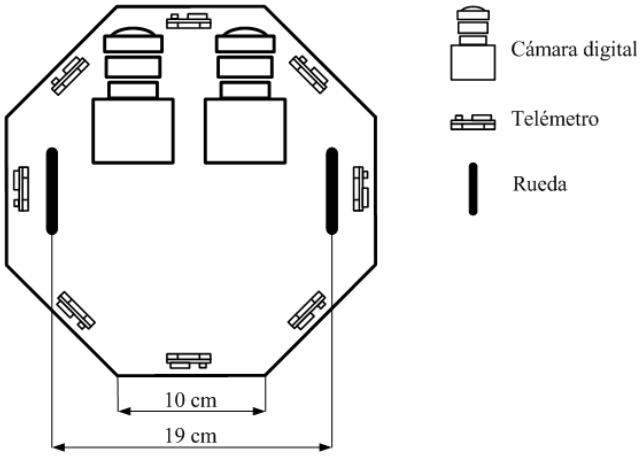


Figura 6.1: Esquema de los sensores, las cámaras y actuadores del robot FemBot.

6.1. Tienen un diámetro de 45 mm. Alrededor de cada rueda se colocan arandelas de goma para mejorar la tracción y evitar que patinen. Estas ruedas son accionadas por motores que permiten moverlas hasta una velocidad máxima de 50 cm/seg. A cada motor se le puede fijar una velocidad con valores entre -127 y 127. El perfil de aceleración es programable (200 cm/seg^2 por defecto). Dadas las características de las ruedas, el robot puede moverse en dirección adelate/atrás, describir arcos y girar en el lugar. Además, el FenBot cuenta con dos encoders incrementales [1] montados en los motores de las ruedas que permiten medir el desplazamiento del robot.

Software de control

Para manejar las distintas funcionalidades del FenBot se tuvieron que desarrollar los drivers correspondientes. Así, se implementaron los drivers para controlar los motores de las ruedas, los sensores de distancia y la cámara de video. A partir de estos controladores de desarrolló la interface de control que se implementó en una librería. Esta librería es la que utiliza la aplicación de aprendizaje para controlar el robot. Para desarrollar los drivers y la librería de control se utilizó el lenguaje de programación `ANSI C`.

A continuación mostramos las funciones de la interface de control y una pequeña descripción de las mismas:

La función `initialize` inicializa los distintos componentes del robot (sen-

[1] Los encoders son sensores que permiten medir la rotación de las ruedas. Existen dos tipos básicos de encoders: incrementales y absolutos. Los incrementales miden la velocidad rotacional del motor (en pulsos) de la cual puede obtenerse el desplazamiento de la posición del robot.

```
int initialize(void)
int getImage(const char*)
int move(float)
int turnLeft(float)
int turnRight(float)
int getBatteryLevel(int[])
int getIRValues(int [])
void done(void)
```

Figura 6.2: Interfaz de control desarrollada para el robot FenBot.

sores, cámara, motores) para que pueda comenzar a operar. `getImage` captura una imagen de 640×480 pixeles en escala de grises y la almacena en un archivo con el nombre que se pasa como parámetro. Para guardar la imagen se utiliza el formato de archivo PGM (Portable Gray Map). La función `move` toma como parámetro la distancia (en cm) hacia adelante (positivos) o hacia atrás (negativo) en línea recta que tiene que moverse el robot. Teniendo en cuenta la curva de aceleración y desaceleración, se calcula durante cuanto tiempo tiene que aplicarse una determinada velocidad a ambos motores para recorrer la distancia pasada como parámetro. Las funciones `turnLeft` y `turnRight` toman como parámetro la cantidad de grados (en radianes) que tiene que girar el robot (en el lugar) a izquierda o derecha respectivamente. La función `getBatteryLevel` recibe un vector de 2 posiciones enteras y devuelve allí los valores de carga de la batería. La función `getIRValues` toma un vector de 8 posiciones y devuelve el valor de los sensores de distancia infrarrojos. Por último `done` apaga los distintos dispositivos para terminar de operar con el robot. Todas las funciones (salvo `done`) devuelven un código de error que informa si la operación pudo ser realizada con éxito, y en caso contrario, un código que especifica el tipo de error para que pueda ser tratado.

6.1.2. El ambiente

El ambiente para las experiencias con el robot Fenbot sigue las mismas especificaciones que las experiencias de simulación. Se cuenta con un pasillo de 1 metro de ancho y 4 metros de largo. Al final del pasillo se encuentra una pared con una puerta centrada, de 40 cm de ancho y 50 cm de altura. Las paredes son de 1 metro de altura. Para la construcción del pasillo se utilizaron placas de telgopor forradas con papel afiche. La puerta se simuló con un papel afiche de color distinto a la pared. Entre las paredes y el piso se colocaron zócalos.

A continuación mostramos algunas fotos del ambiente y del robot FenBot:

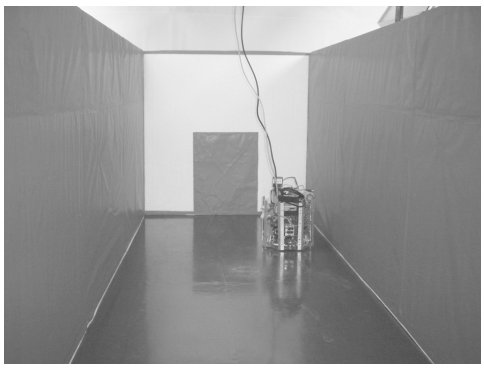


Figura 6.3: El robot FenBot en el pasillo.

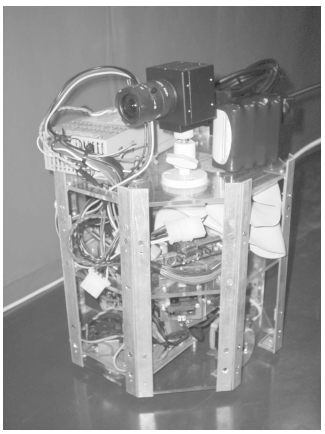


Figura 6.4: Robot FenBot.

6.1.3. Ajuste de parámetros para el procesamiento de las imágenes

En la sección 4.4 describimos el procedimiento seguido para obtener las rectas a partir de las imágenes capturadas del entorno, y describimos los parámetros que deben ser ajustados. A continuación mostramos los valores de estos parámetros utilizados para las experiencias con el robot FenBot. Estos valores fueron encontrados de manera experimental, siguiendo lo explicado en la sección 4.4.

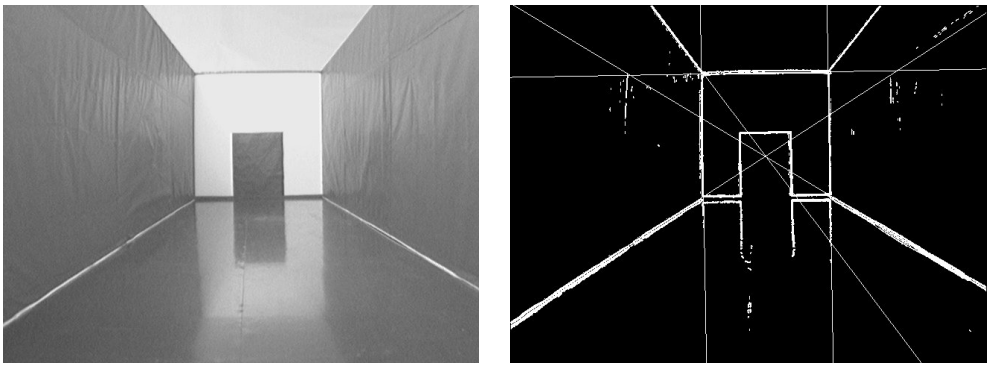
Tomar un ρ por pixel y un θ por grado en una imagen de 640×480 pixels nos da 180 valores de θ diferentes y $diag = \lceil \sqrt{640^2 + 480^2} \rceil = 800$ valores de ρ (ver 3.2), definiendo una matriz de acumuladores H de 180 columnas y 800 filas (ver 4.4.1). Como vimos en 4.4.2, la definición del tamaño de la vecindad depende de las dimensiones de la matriz H. Experimentalmente se notó que una vecindad que fuera de un séptimo de la cantidad de filas y columnas de la matriz H es apropiada para las experiencias con el robot real.

Etapa	Parámetro	Valor
Detección de bordes (umbralización)	t_0	90
	t_1	130
Transformada de Hough	ρ	uno por pixel
	θ	uno por grado
Detección de rectas (vecindad y umbralización)	n	$\#filas(H)/7$
	m	$\#cols(H)/7$
	u_r	70
	c	0,25

Cuadro 6.1: Parámetros del procedimiento seguido para obtener las rectas a partir de las imágenes obtenidas con el FenBot.

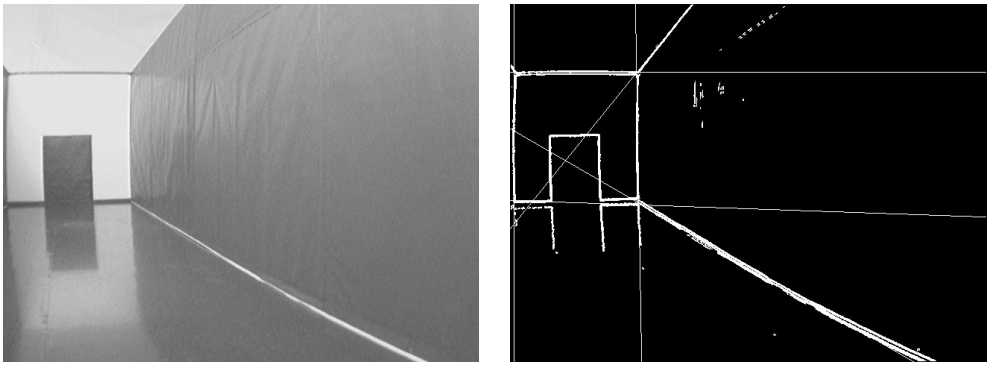
Por lo tanto, en las experiencias en simulación se adoptó: $n = \lceil 800/10 \rceil = 80$ y $m = \lceil 180/7 \rceil = 26$. Además, para la detección de rectas fijamos el umbral absoluto u_r en 70 y el factor c para el umbral relativo en 0,25.

A continuación mostramos algunas imágenes capturadas con la cámara del FenBot y la imagen procesada asociada:



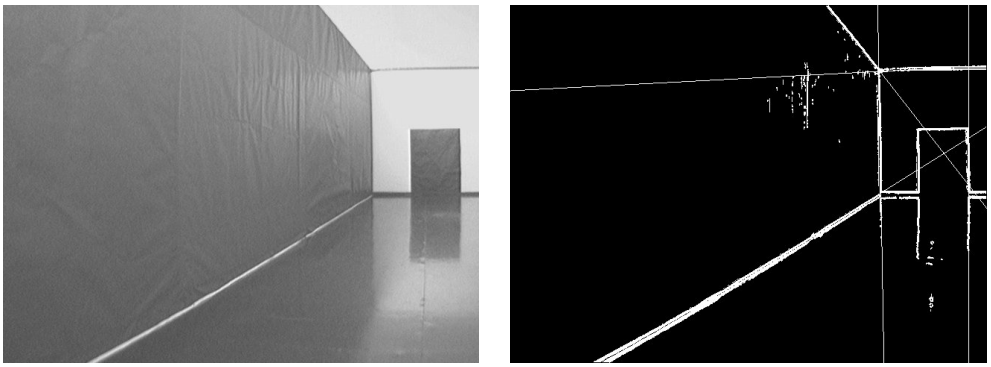
a) b)

Figura 6.5: a) Imagen capturada desde el inicio del pasillo y con 0 grados de orientación. b) Imagen procesada.



a) b)

Figura 6.6: a) Imagen capturada desde el inicio del pasillo y con 20 grados a la derecha de orientación. b) Imagen procesada.



a) b)

Figura 6.7: a) Imagen capturada desde el inicio del pasillo y con 20 grados a la izquierda de orientación. b) Imagen procesada.

6.1.4. Implementación de la función de refuerzo

Cálculo del posicionamiento momentáneo

La función de refuerzo utilizada requiere conocer la distancia relativa entre el robot y el objetivo (el fin del pasillo). La posición del objetivo es conocida, pero la posición del robot debe ser calculada. En el caso de las experiencias en simulación esto no es un mayor problema porque podemos obtener las posiciones absolutas del robot a partir de la librería `java3d`. Pero en el caso de las experiencias con el FenBot debemos utilizar algún método para estimar la posición del robot en cada momento. Una solución posible para este problema es utilizar la técnica de odometría. Esta técnica consiste en el cálculo de ecuaciones geométricas simples a partir de la información suministrada por los encoders. A partir de las ecuaciones odométricas podemos traducir la cantidad de vueltas que dan las ruedas (medidas en pulsos por los encoders), en un desplazamiento del robot relativo al piso.

Sin embargo, para el presente trabajo se utilizó otro método. Como se cuenta con los perfiles de aceleración y desaceleración de los motores[29], se puede calcular el tiempo que se tiene que aplicar una velocidad dada a ambos motores para obtener un desplazamiento determinado. De la misma forma, podemos calcular cuanto tiempo se tiene que aplicar una velocidad a un motor y su opuesta al otro, para obtener un giro (a izquierda o derecha) de un determinado ángulo. De esta manera, y utilizando ecuaciones trigonométricas simples, podemos estimar la posición del robot luego de cada acción. Queda claro que estos son cálculos teóricos que no necesariamente tienen que coincidir exactamente con el desplazamiento (o giro) real que experimenta el robot. Sin embargo, las experiencias realizadas mostraron que este método arroja una aproximación al desplazamiento real lo suficientemente buena como para poder ser utilizada para calcular el valor de la función de refuerzo en cada momento. Además, la degradación que sufre la estimación a partir de la acumulación de errores (o diferencias) entre la posición calculada y la real se elimina cuando se reinicia un nuevo episodio. En ese momento el robot es colocado en una posición conocida por un operador y se actualizan las variables de posición del robot.

Reflejos

La función de refuerzo tiene dos tipos de reflejos: uno cuando el robot detecta una o ninguna recta y otro cuando choca con una pared. El primer reflejo se implementa fácilmente a partir del resultado del procesamiento de cada imagen. En el caso del segundo tipo de reflejo se utilizaron los telémetros para determinar cuando el robot choca. No se requiere conocer la distancia a la que se encuentra el robot de la pared sino simplemente cuando está lo suficientemente cerca para considerar que chocó. Para eso se fijó un umbral a partir del cual se considera que ese sensor está activado. De esta forma cada sensor tiene sólo dos estados: activado o no activado. Cada vez que se realiza una acción se miden los valores de los telémetros y se determina cuales están activos y cuales no. Si hay algún algún sensor activo consideramos que el robot chocó y entonces se activa el reflejo.

Como dijimos en la sección 6.1.1 la sensibilidad de los telémetros puede verse afectada por condiciones externas, como por ejemplo por una fuente de luz que afecte al receptor. Eso se traduce en que a veces los sensores arrojan falsos valores y podemos pensar que está activado cuando en realidad no lo está. Para atacar este problema, cada vez que un sensor supera el umbral de activación, volvemos a sensar para ratificar el valor. Recién cuando dos veces consecutivas los valores de un sensor superan el umbral consideramos que está activado. De esta forma superamos el problema de las falsas activaciones.

6.2. Resultados

Consideraciones previas

Para definir qué experiencias realizar en el robot real, utilizamos los resultados de las experiencias simuladas. Como vimos en 5.2.7, las políticas aprendidas utilizando 3 rectas tienen resultados satisfactorios a partir de 3000 iteraciones al utilizar un ϵ que decae a $0,05$ en 6.000 iteraciones. En particular, podemos observar en la figura 5.9 que la política aprendida utilizando 3 rectas, 5 buckets de ρ y 5 buckets de θ tiene un 93 % de episodios exitosos en 3.000 iteraciones. Aparte, esta configuración de parámetros es la que determina la menor cantidad de estados, como se puede observar en la figura 5.5. Por estas razones, utilizamos la configuración de 3 rectas, 5 buckets de ρ y 5 buckets de θ con un ϵ que decae a $0,05$ en 6.000 iteraciones.

Para testear la política obtenida utilizamos posiciones iniciales aleatorias. En principio, deberíamos utilizar las 180 posiciones iniciales que son muestra para las pruebas que realizamos en simulación. Sin embargo, como el costo de cada prueba en el robot real es considerable (en el sentido de tiempo, atención de un operario, tiempo de carga de batería, etc), quisimos minimizar la cantidad de posiciones iniciales a utilizar. Como vimos en 5.2.7, para determinar el tamaño de la muestra utilizamos la configuración de mayor cantidad de estados (4 rectas, 10 buckets de ρ y 10 buckets de θ). Este tamaño se mantuvo en todas las experiencias para poder comparar los resultados. Considerando que con una configuración de 4 rectas, 10 buckets de ρ y 10 buckets de θ tenemos 728 estados y que con 3 rectas, 5 buckets de ρ y 5 buckets de θ tenemos 152 estados, podemos esperar que la cantidad de posiciones iniciales necesaria para estabilizar una muestra sea menor. En la figura 6.8 podemos ver la prueba de estabilidad para la política aprendida con 3 rectas, 5 buckets de ρ y 5 buckets de θ a 6.000 iteraciones en simulación.

Teniendo en cuenta estos resultados, decidimos testear la política aprendida en el robot con 55 posiciones iniciales, incluídas dentro de la muestra de 180 posiciones iniciales utilizadas para testear las políticas en simulación. Si bien disminuir el tamaño de la muestra para testear la política puede acarrear algún tipo error, es de esperar que este error sea pequeño respecto al porcentaje real de episodios exitosos, en función de las variaciones que se observan en la figura 6.8.

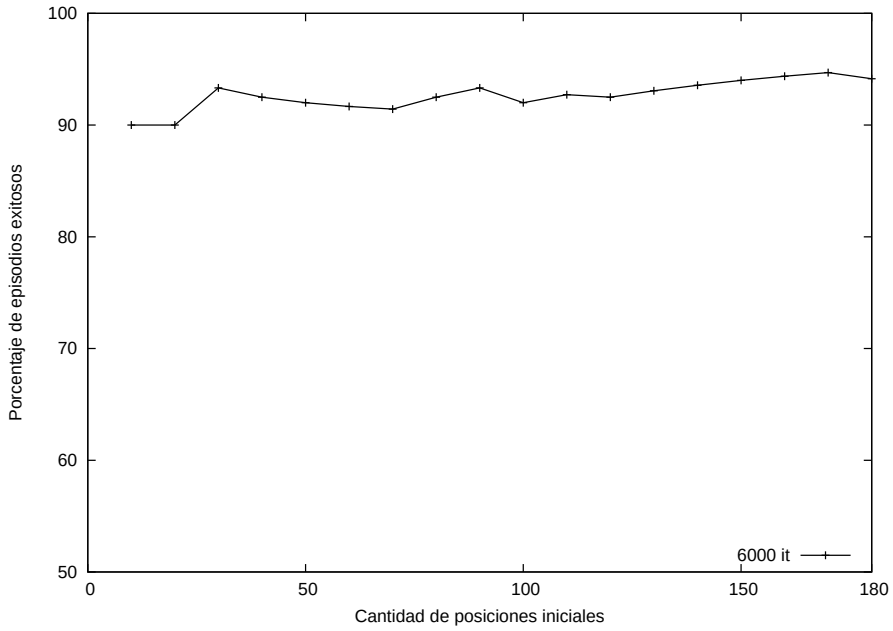


Figura 6.8: Porcentaje de episodios exitosos, para distintas cantidades de posiciones iniciales aleatorias para la política aprendida con 3 rectas, 5 buckets de ρ y 5 buckets de θ a 6000 iteraciones en simulación.

Resultados de la política

La experiencia realizada en el robot FenBot tiene como parámetros 3 rectas, 5 buckets de ρ y 5 buckets de θ con un ϵ que decae por debajo de $0,05$ en 6.000 iteraciones. Testeamos las políticas obtenidas con 55 posiciones iniciales aleatorias que son parte de las 180 posiciones iniciales aleatorias utilizadas para testear las políticas en simulación. A continuación mostramos los resultados correspondientes a las políticas obtenidas luego de 3.000, 4.000 y 5.000 iteraciones de aprendizaje.

Cantidad de iteraciones	3.000	4.000	5.000
Episodios exitosos	53,57 %	72,42 %	92, 72 %
Estados no visitados	12, 14 %	10, 98 %	9, 83 %

Cuadro 6.2: Resultados para las políticas aprendidas con la configuración de 3 rectas, 5 buckets de ρ y 5 buckets de θ a 3000, 4000 y 5000 iteraciones.

Una observación interesante es que la cantidad de estados en el entorno real es mayor a la cantidad de estados en el entorno simulado. Con 3 rectas,

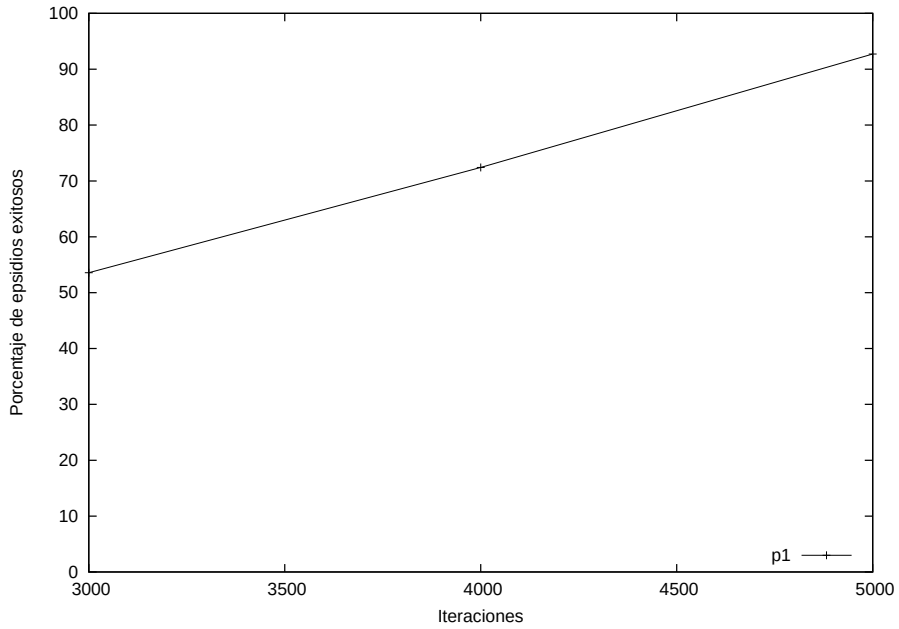


Figura 6.9: Porcentaje de episodios exitosos en función de la cantidad de iteraciones.

5 buckets de ρ y 5 buckets de θ tenemos 152 estados en la simulación y 172 en el entorno real. Esto puede deberse a que los giros y avances en la realidad no son exactos como en el simulador, y por lo tanto durante el aprendizaje el robot en la realidad pasa por más posiciones diferentes.

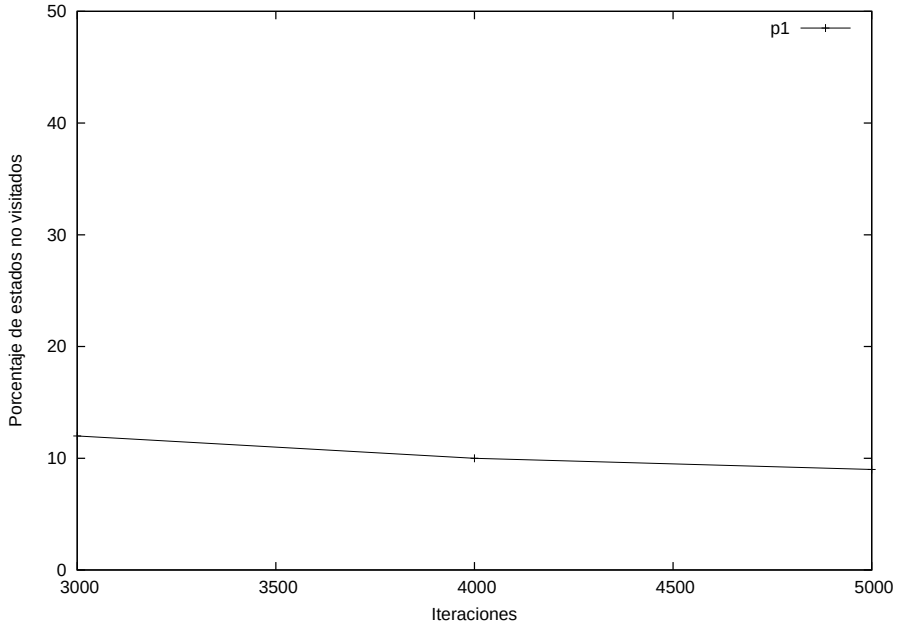


Figura 6.10: Porcentaje de estados que no fueron recorridos durante el aprendizaje en función de la cantidad de iteraciones.

6.3. Conclusiones de las experiencias con el robot FenBot

A partir de los resultados obtenidos en las experiencias con el robot FenBot, podemos concluir que la política aprendida es satisfactoria. Pudimos verificar que el método propuesto aplica en un sistema real para la tarea de recorrer un pasillo simple con un robot móvil.

Las experiencias realizadas en simulación permitieron ajustar los parámetros del método propuesto antes de realizar las experiencias con el robot real, disminuyendo considerablemente el tiempo de las experiencias preliminares. Esto fue posible gracias a que el simulador desarrollado permite alcanzar una simulación bastante ajustada del entorno real. Sin embargo, las políticas obtenidas en simulación no dieron buenos resultados al ser testeadas en el robot real. Una posible razón es que las imágenes obtenidas con el simulador, si bien se aproximan bastante, no son exactamente iguales a las obtenidas con el FenBot. Queda pendiente intentar ajustar el simulador y la cámara del FenBot de forma tal que una política aprendida en simulador pueda ser ejecutada en el robot real.

Las experiencias con el FenBot nos permitieron comparar los resultados obtenidos en simulación con las experiencias reales. Las pruebas con el robot real resultan igualmente satisfactorias que las realizadas en simulación.

Por último, pudimos observar que la cantidad de estados en el entorno real es mayor a la cantidad de estados en el entorno simulado. Esto puede deberse a que los giros y avances en la realidad no son exactos y, por ende, el robot pasa por más posiciones diferentes durante el aprendizaje.

Capítulo 7

Conclusiones

En este trabajo estudiamos la utilización de Aprendizaje por Refuerzo en problemas para los cuales se tiene una percepción basada en visión. Para mapear el espacio de imágenes a una representación cuyo cardinal haga factible el uso de técnicas de Aprendizaje por Refuerzo como Q-Learning propusimos un método que utiliza la transformada lineal de Hough. La hipótesis del trabajo fue verificada experimentalmente con lo cual se puede concluir que una representación basada en una cantidad pequeña de rectas inferidas de las imagenes sensadas, contiene la información necesaria del entorno y reduce el cardinal de estados lo suficiente, como para poder utilizar el algoritmo de Q-Learning, al menos para los casos de estudio considerados. Para la representación de los estados se utilizó la forma normal de la rectas que involucra los parámetros ρ y θ. Por eso, el método propuesto tiene tres parámetros para la definición de los estados: la cantidad de rectas, la cantidad de buckets de ρ y la cantidad de bucktes de θ. Además, como parte del método se propuso una técnica de exploración dirigida libre de modelo.

La primera etapa del trabajo consistió en implementar una librería para el procesamiento de la imágenes. Para la detección de bordes, probamos varios operadores de gradiente y técnicas de umbralización. Concluimos que la mejor combinación para la detección de bordes, como preprocesamiento de las imágenes para encontrar rectas utilizando la transformada de Hough, es la aplicación del operador de Sobel y la umbralización con histéresis. Para fijar la granularidad de los parámetros de Hough probamos con varios valores. Pudimos observar que la implementación lograda es lo suficientemente rápida como para utilizar una granularidad muy fina (un grado y un pixel). El procedimiento implementado para obtener las rectas de las imágenes capturadas funciona tanto en simulación como con la cámara de video del robot FenBot y requiere ajustar pocos parámetros, lo cual hace que sea muy versátil.

Desarrollamos un entorno de simulación para las primeras experiencias antes de realizar las experiencias con el robot real. Con las experiencias en simula-

ción pudimos validar el método propuesto para un caso de estudio simple, que involucra un agente móvil en un ambiente sencillo. Luego, pusimos a prueba el método propuesto en un ambiente donde una percepción del entorno basada en otros sensores, como los sensores de proximidad, no son suficientes. En ambos casos, los resultados fueron muy satisfactorios. Se comprobó que un número pequeño de rectas (3 o 4) es suficiente para obtener una representación del ambiente apropiada para utilizar el algortimo de Q-Learning. Además, pudimos observar que la cantidad de estados reales es mucho menor que la cantidad de estados matemáticamente posibles y no escala exponencialemnte respecto de la cantidad de rectas, de buckets de ρ o de θ.

Luego, aplicamos el método con el robot móvil FenBot. Pudimos verificar que el método propuesto funciona en un sistema real. Las experiencias con el FenBot nos permitieron comparar los resultados obtenidos en simulación con las expericias reales. Pudimos observar que las pruebas con el robot real resultan igualmente satisfactorias que las realizadas en simulación. La cantidad de estados en las experiencias con el robot real es más de un 12 % que la cantidad de estados en simulación.

En cuanto al diseño de la función de refuerzo, podemos concluir que los estimadores de progreso fueron un factor determinante en la tarea a realizar. Las experiencias preliminares que realizamos sin los estimadores de progreso fracasaron.

La técnica de exploración dirigida propuesta no requiere conocer la dinámica del entorno, es decir, es libre de modelo. Esta característica hace que sea apropiada para utilizar en la etapa de aprendizaje del método propuesto. Pudimos observar que la utilización de esta técnica disminuye considerablemente la cantidad de iteraciones necesarias para aprender una política satisfactoria. Esta técnica puede ser aplicada en el desarrollo de otros comportamientos de robots autónomos que utilicen AR. Sería interesante poder comparar la técnica porpuesta con la $\epsilon - greedy$, en comportamientos que involucren otros sensores en lugar de una cámara de video. Otro estudio interesante resultaría de comparar la técnica de exploración propuesta con la técnica counter-based en problemas donde se conoce la dinámica del entorno. Por último, se podría incorporar a la técnica propuesta un factor de decaimiento (que multiplica los contadores en cada iteración) para tener en cuenta el momento en que se elige una determinada acción y darle más relevancia a las que hace más tiempo que no se ejecutan.

El método propuesto soluciona las dificultades que surgen del uso de AR para el problema de seguir un pasillo simple o cruzado. Es libre de modelo y tiene pocos parámetros que ajustar, lo cual hace pensar en que puede aplcarse a otros problemas de AR que involucren una percepción del entorno basada en visión. Por eso, resultaría importante continuar el presente trabajo aplicando el método propuesto en ambientes más complejos y/o que involucren otro tipo de tareas.

Bibliografía

[1] R. C. Arkin, *Behavior-Based Robotics*, MIT Press, Cambridge, 1998.

[2] J.M. Santos y C. F. Touzet, Q-learning and Robotics, *European Simulation Symposium 2001*, Marseille, 2001.

[3] R. Sutton, *Reinforcement Learning, An introduction*, A Bradford Book, MIT Press, Cambridge, 1998.

[4] C.J.C.H. Watkins, Learning from Delayed Rewards, PhD thesis, King's College, Cambridge, 1989.

[5] F. Fernández Rebollo, Aprendizaje por Refuerzo en espacios de estados continuos, Tesis Doctoral, Escuela Politécnica Superior, Universidad Carlos III de Madrid, Leganés, 2002

[6] C.J.C.H. Watkins y P. Dayan, Q-Learning, Technical Note, *Machine Learning*, Volumen 8, Número 3, pag. 279–292, 1992.

[7] R. Sim y G. Dudek, Learning and evaluating visual features for pose estimation, *Proc. of the Seventh International Conference on Computer Vision (ICCV'99)*, Kekyra, Greece, Sept 1999.

[8] S. Se, D. Lowe y J. Little, Mobile Robot Localization and Mapping with Uncertainty using Scale-Invariant Visual Landmarks,*International Journal of Robotics Research*, Volumen 21, Número 8, pag. 735-758, Agosto 2002.

[9] S. Se, D. Lowe y J. Little, Vision-based Global Localization and Mapping for Mobile Robots,*IEEE Transactions on Robotics*, Volumen 21, Número 3, pag 364-375, Junio 2005.

[10] J. M. Santos, Contribution to the study and the design of reinforcement function, P.H.D Tesis, Universidad de Buenos Aires, 1999.

[11] Milan Sonka, Vaclav Hlavac y Roger Boyle, *Image Processing, Analysis, and Machine Vision*, ITP, PWS. Publishing, 2° edition, 1998.

[12] T. Martínez-Marín y T. Duckett, Fast Reinforcement Learning for Vision-Guided mobile robots, *Proc. of the 2005 IEEE International Conference on Robotics and Automation*, Barcelona, España, 2005

[13] C. Distante, A. Anglani y F. Taurisano, Target Reaching by Using Visual Information and Q-Learning Controllers, *Autonomous Robots*, Volume 9, Número 1, pag 41-50, Agosto 2000.

[14] S. B. Thrun, The Role of Exploration in Learning Control, en *Handbook of Intelligent Control: Neural, Fuzzy and Adaptative Approaches*, D. A.White y D. A. Sofge editores, Van Nostrand Reinhold, Nueva York, 1992.

[15] S. B. Thrun, Efficient Exploration in Reinforcement Learning, Technical Report, School of Computer Science, Carnegie-Mellon University, 1992.

[16] A. Bonarini, C. Bonancina y M. Matteucci, An approach to the design of reinforcement functions in real world, agent-based applications, *Systems, Man and Cybernetics, Part B, IEEE Transactions*, 2001

[17] Maja J Mataric, Reward Functions for Accelerated Learning, *Machine Learning: Proceedings of the Eleventh International Conference*, New Brunswick, New Jersey, 10-13 Julio 1994.

[18] C. W. Anderson, Q-Learning with Hidden-Unit Restarting, *Advances in Neural Information Processing System 5*, Morgan Kaufman Publishers, pag 81-88, 1993.

[19] L. Lin, Reinforcement Learning for Robots Using Neural Networks, PhD thesis, Carnegie-Mellon University, 1993.

[20] D. Randall et al, *Eckert - Fisiología Animal*, Editorial Interamericana - Mac Grow Hill, 1998.

[21] S. Florczyk, *Robot Vision - Video-based Indoor Exploration with Autonomous and Mobile Robots*, Wiley-VCH, 2005.

[22] B. Klaus y P. Horn, *Robot Vision*, MIT Press, 1986.

[23] P. V. C. Hough, Method and means for recognizing complex patterns, U.S. Patent 3069654, 1962.

[24] R. O. Duda y P. E. Hart , Use of the Hough Transformation to Detect Lines and Curves in Pictures, *Comm. ACM*, Volumen 15, Número 1, 1972

[25] D. Ziou y S. Tabbone, Edge Detection Techniques - An Overview, *Int. J. Pattern Recognition Image Anal.*, Volumen 8, pags 537-559, 1998.

[26] R. C. Gonzalez y R. E. Woods, *Digital Image Processing*, Prentice Hall, Segunda Edicción, 2002.

[27] D.H. Ballard, Generalizing the Hough transform to detect arbitrary shapes, *Pattern Recognition* Volumen 13, pag 111-122, 1981.

[28] SHARP *Distance Measuring Sensors*, General Application Note, 2003

[29] P. Borensztejn, A. Stoliar, J. M. Santos, J. Jacobo Berlles Julio, Velocity profile for fast and precise mobile robot using step motor actuators. *Proceeding of the FIRA Robot World Congress 2003*, Viena, Austria, October 1–3, 2003.