Universidad Zaragoza
1542

Trabajo Fin de Grado

Aprendizaje por Refuerzo y Planificación en un Sistema Multirobot

Reinforcement Learning and Planning on a Multi-Robot System

Autor/es

Alberto San Miguel Tello

Director/es

Cristian Florentin Mahulea
Javier Civera Sancho

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2017

Trabajo Fin de Grado

Aprendizaje por Refuerzo y Planificación en un Sistema Multirobot

Alberto San Miguel Tello

Resumen

Este Trabajo Fin de Grado aborda el problema de navegación de unos agentes móviles sobre un entorno con obstáculos desconocidos para ellos, realizando los movimientos necesarios para alcanzar una serie de posiciones finales partiendo de unas iniciales. El método elegido para su resolución ha sido el algoritmo *Dyna-Q*, que combina técnicas Aprendizaje por Refuerzo y Planificación, implementándose primero en el caso de un agente para después extenderlo al caso multiagente mediante un planteamiento propuesto en este estudio. Se ha realizado una fase de experimentación sobre los parámetros del algoritmo para el caso de un agente con el fin de interpretar su influencia sobre la ejecución del problema, y determinados resultados se han aplicado sobre los casos multiagente y se ha evaluado su viabilidad. Finalmente, se ha planteado la implementación de los métodos y evaluación de los resultados sobre una plataforma real compuesta por varios agentes móviles, abarcando las posibles limitaciones que presente.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Alberto San Miguel Tello,

con nº de DNI 17769667M en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Aprendizaje por Refuerzo y Planificación en un Sistema Multirobot,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 23 de Junio de 2017

Fdo: Alberto San Miguel Tello

Índice

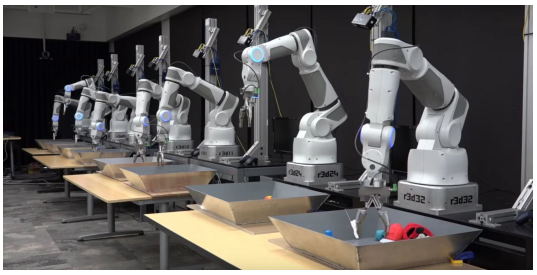
Índice de figuras

Índice de tablas

1. Introducción

Hoy en día una gran parte de los esfuerzos en el ámbito de la Ingeniería se enfocan tanto hacia la automatización de procesos sencillos y repetitivos como de tareas que requieran un entendimiento y manipulación del entorno complejos, utilizando para ello sistemas robóticos (de aquí en adelante, agentes). Muchas de estas técnicas se engloban dentro de lo que se conoce como Inteligencia Artificial (IA), que tiene como objetivo dotar a dichos agentes de capacidades cognitivas propias de los seres racionales, como es la percepción y la toma de decisiones dentro de un entorno.

(a) Juego *Breakout* de Atari 2600



(b) Brazos robóticos de la UC en Berkeley y Google

Figura 1: Ejemplos de aplicación de métodos de Aprendizaje por Refuerzo

Dentro de la IA existe una diferenciación entre los procesos en los que al agente adquiere esas capacidades mediante comparación con casos similares que se le han proporcionado, o aquellos en los que el agente es el que a través de actuar directamente sobre el entorno, obtiene una experiencia que le permite adquirir esas capacidades. Este último se denomina como Aprendizaje por Refuerzo es especialmente útil en casos donde la tarea es difícil de tipificar en estados diferenciados, de manera que un aprendizaje activo se presenta como una alternativa muy eficaz. Prueba de ello son los numerosos ejemplos existentes, como es el método DQN planteado por la empresa DeepMind, propiedad de Google, que une el algoritmo *Q-learning*, uno de los más importantes dentro del Aprendizaje por Refuerzo, con una Red Neuronal profunda para la resolución de problemas donde se pretende que un único agente domine de forma simultánea una serie de tareas totalmente distintas, al igual que lo haría un ser humano. Se demostró así que un agente fue capaz de dominar mejor que los jugadores humanos expertos una gran variedad de juegos de la consola Atari 2600 [1]. Un ejemplo más centrado en una aplicación real sería el experimento llevado a cabo en la Universidad de California en Berkeley y con la colaboración de Google, en el que se utilizaron alrededor

de 200 brazos robóticos conectados entre sí, que mediante Aprendizaje por Refuerzo interaccionaban de forma directa con una serie de objetos con el fin de obtener la experiencia suficiente que les permitiera hallar la forma más adecuada de agarrarlos [2].

También cabe destacar que parte de la dificultad que presentan las técnicas de IA recae en la necesidad de adaptarlas a la naturaleza particular del caso en el que se aplique o de salvar algunas limitaciones que presente de cara a su funcionamiento en sistemas físicos. Por este motivo, la fase de determinación y diseño de los parámetros del algoritmo en cuestión tiene un gran peso dentro del planteamiento del problema, y la experimentación cobra sentido para conocer como afectan en el proceso.

Por otro lado, los Sistemas Multiagente (SMA) también han sido recientemente objeto de estudio en los últimos años y están fuertemente ligados a la IA, al trabajar sobre la interacción de los agentes de forma coordinada en la resolución de problemas de mayor complejidad, o en aquellos donde la participación de varios sistemas inteligentes sea estrictamente necesaria. Su aspecto fundamental es la descentralización, de manera que los agentes son en gran parte autónomos, influyendo en cierta medida la interacción con los demás en su comportamiento. Como ejemplo reciente, tendríamos la plataforma desarrollada por Intel y compuesta por 100 drones que actúan de forma coordinada, y que ya se han utilizado con fines lúdicos en la formación de patrones de luces en el aire[1]. Los SMA también van más allá de la Ingeniería, basándose por ejemplo algunas de sus técnicas en comportamientos de coordinación presentes en los seres vivos, o siendo muchos de los métodos estudiados útiles en el modelado de estructuras sociales o de mercados económicos.

La situación que se estudia es un problema clásico: trazar una ruta desde un punto de partida hasta un destino dentro de un mapa rígido. Este problema adquiere gran relevancia, no sólo por los numerosos casos reales de navegación en los que se aplica, sino porque sus métodos son fácilmente extrapolables a problemas donde exista un punto inicial y otro final, pasando por una serie de situaciones intermedias que definen la "ruta". En lo que concierne a este estudio, el problema se planteará de forma extendida, de manera que un cierto número de agentes móviles (robots) tendrán la tarea de alcanzar unos destinos, partiendo de unas salidas y dentro de un mapa. El mapa tendrá unos límites definidos y dentro de él existirán un cierto número de obstáculos fijos. Asumiendo por parte de los agentes un conocimiento

[1]En este vídeo se recogen algunas pruebas realizadas en el proceso de extensión de la plataforma a 500 drones: *https://www.youtube.com/watch?v=aOd4-T_p5fA*

previo del mapa al completo, la resolución se puede abordar de múltiples formas, como por ejemplo mediante algoritmos de búsqueda avanzados, redes de Petri o métodos de Aprendizaje por Refuerzo. Pero precisamente son éstos últimos los que se presentan como los más idóneos en los casos en los que se desconoce de forma parcial o total el mapa y/o su dinámica, siendo por tanto necesaria la interacción directa con el entorno. De esta forma el problema se enmarcará en una situación donde el modelo de los agentes no es completo y/o es erróneo, más cercana por lo general a un caso real, y se va a enfocar utilizando la arquitectura *Dyna-Q*, fundamentada en el mencionado algoritmo *Q-learning*.

En líneas generales, el alcance de lo realizado dentro de este trabajo se puede particularizar en los siguientes objetivos:

- Adaptación del algoritmo *Dyna-Q* y definición de sus parámetros de acuerdo al planteamiento del problema.
- Extensión del algoritmo para su implementación en casos multiagente, mediante un planteamiento desarrollado de forma particular para este estudio.
- Evaluación de la influencia de los parámetros en el caso de un agente mediante experimentación e interpretación de los resultados obtenidos con el fin de estudiar su aplicación en el caso multiagente.
- Implementación tanto del algoritmo *Dyna-Q* como de la extensión al caso multiagente planteada en una plataforma con robots móviles reales, adaptándose a sus condicionantes.

2. Algoritmo *Dyna-Q*

Los métodos que exploran el concepto de dotar a los agentes de la capacidad de tomar de decisiones para encaminarse hacia un objetivo se fundamentan en la idea de orientar dado un estado determinado s la toma de una acción a, pasando a otro nuevo estado s' dentro de un entorno. Éste último responderá al agente mediante señales que le indiquen en qué medida se comporta hacia o en contra del objetivo que busca. Dichas señales son conocidas como recompensas r, y el agente buscará maximizarlas como forma de alcanzar el objetivo.

A nivel interno, la toma de decisiones se manifiesta en forma de una política π, que determinará el comportamiento del agente a lo largo de la tarea. Para evaluar dicha política es necesario observar cuál será la recompensa acumulada esperada al seguirla, y así se definen unas funciones dependientes de s, s' y a. El valor de tomar una acción a en un determinado estado s siguiendo una política π se puede formular tal y como se desarrolla en (1), asumiendo un Proceso de Decisión de Markov (MDP) [3] y un entorno determinista:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi \left[\sum_{k=0}^{n} \gamma^k \; R_{t+k+1} \; | S_t = s, A_t = a \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma \; \sum_{k=0}^{n} \gamma^k \; R_{t+k+2} \; | S_t = s, A_t = a \right] \\ &= r(s,a,s') + \gamma \mathbb{E}_\pi \left[\sum_{k=0}^{n} \gamma^k \; R_{t+k+2} \; | S_{t+1} = s', A_{t+1} = a' \right] \\ &= r(s,a,s') + \gamma \; q_\pi(s',a') \end{aligned} \tag{1}$$

donde el valor de $q_\pi(s,a)$ es la esperanza de las recompensas acumuladas a lo largo de un número n de intervalos de tiempo, tal que en el intervalo actual t se tome la acción a dentro del estado s. La recompensa acumulada se expresa como el sumatorio de los productos de la recompensas R en los estados futuros y el factor de descuento γ. Este factor toma valores entre 0 y 1, y al elevarlo al intervalo futuro k marca el valor que representa en t la recompensa futura. Esta esperanza se desarrolla como la recompensa esperada $r(s,a,s')$ al ejecutar la acción a en el estado actual, más la esperanza de las recompensas acumuladas en el estado siguiente. La expresión final obtenida se conoce

como la Ecuación de Bellman para $q_\pi(s, a)$. De esta forma, el agente tiene constancia de las repercusiones de su política y por tanto es capaz de enfocar sus acciones hacia aquella que maximice su recompensa acumulada, es decir, hacia aquellas acciones que maximicen los valores de $q(s, a)$.

Si el agente toma esta consideración, obtendrá la máxima recompensa acumulada posible y consecuentemente su política será la óptima. Esta premisa parte de la certeza de que los valores que toma $q(s, a)$ son los "reales", es decir, aquellos que describen totalmente las repercusiones futuras de tomar una acción a en un estado s. Suponiendo que el agente carece de experiencia alguna sobre el entorno, necesitará por tanto interaccionar con él para poder alcanzar dichos valores. Por lo tanto, se puede justificar el proceso de aprendizaje como una interacción entre el entorno y el agente, en el que éste último evalúa las repercusiones de sus acciones y conforme a ellas orienta su política hacia la que le permite obtener el objetivo deseado de forma óptima.

2.1. Aprendizaje por Refuerzo y Planificación

Si bien es cierto que el agente (por lo general) carece de experiencia alguna, como paso previo al aprendizaje puede partir de un modelo que predice la respuesta del entorno ante sus acciones bajo unas condiciones. Este modelo asocia los posibles estados alcanzables dadas las posibles acciones en cada uno de ellos, así como el sistema de recompensas que conlleva.

El modelo, por tanto, puede ser utilizado por el agente para evaluar el efecto de sus acciones. Al tener una predicción del comportamiento del entorno, se puede generar experiencia simulada que permita evaluar una política y mejorarla. Tal y como se ha expuesto anteriormente, la convergencia de estos valores a los reales asegura la convergencia a una política óptima. Este procedimiento es denominado **Planificación** (*Planning*), y su ventaja yace en problemas donde el espacio de estados y acciones requeriría de un proceso de aprendizaje muy exhaustivo como para que se base únicamente en experiencia generada por la interacción real con el entorno.

La principal desventaja de la Planificación es precisamente que recae exclusivamente en el modelo. Para que la experiencia sea útil en el proceso de aprendizaje debe emplearse aquel modelo que represente de forma exacta la dinámica del entorno. En algunos casos es factible obtener uno que refleje a la perfección su comportamiento, pero con entornos cambiantes o donde se desconoce o se ha estimado de forma incorrecta su respuesta bajo unas condiciones, el modelo es probable que caiga en predicciones erróneas que

previsiblemente alejarían al agente de obtener la política óptima. Por esta razón, en estos últimos casos (que suelen ser los que se dan en los entornos físicos), para comprobar la corrección del modelo con el que se parte, es necesario tener experiencia real con el entorno. Esto permite mejorar el modelo del que se dispone para que sea más fiel al entorno que predice, ya sea verificándolo o corrigiéndolo; y a la vez evaluar las acciones del agente y sus repercusiones mediante experiencia real, lo que se ha denominado como **Aprendizaje por Refuerzo** (*Reinforcement Learning*).

2.2. Arquitectura *Dyna*

La arquitectura *Dyna* integra Planificación y Aprendizaje por Refuerzo de forma simultánea durante el proceso de aprendizaje. Las experiencias simulada y real se emplean en la evaluación y mejora de la política del agente, y ésta última además se utiliza para actualizar el modelo en base a la realidad experimentada. De esta forma se obtiene una experiencia simulada que se acerca a la real, y por tanto es de mayor utilidad para el agente. Estas relaciones se han representado de una forma gráfica en la Figura 2

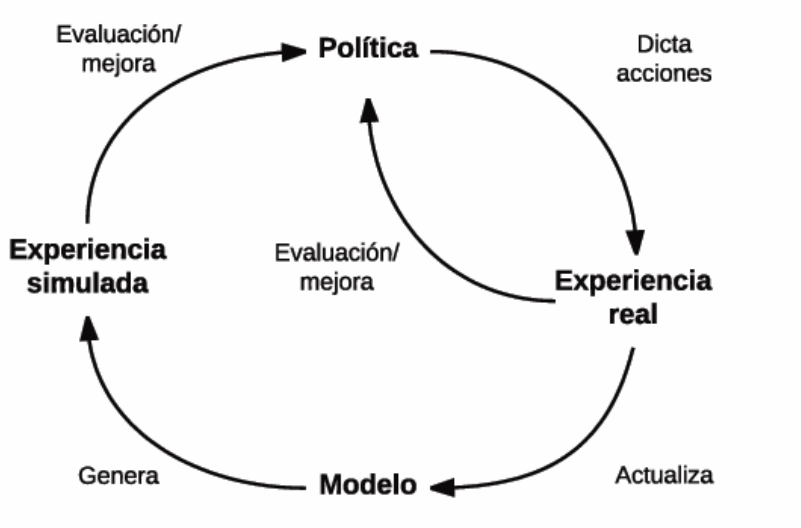


Figura 2: Relaciones entre procesos de Planificación y Aprendizaje por Refuerzo

En este estudio se va a trabajar sobre la estructura *Dyna* que utiliza el *Q-learning*, dando lugar al algoritmo *Dyna-Q*. Su implementación en sistemas computacionales hace necesario que ambos procesos se den en serie y no en paralelo. En el Algoritmo 1 se muestra de forma esquemática.

Algoritmo 1 *Dyna-Q*

Inicialización: Matriz $Q(s,a)$ y $Modelo\ (s,a)$ para todo $s \in \mathcal{S}$ y $a \in \mathcal{A}(s)$.
Hasta estado terminal T:

1: $s \leftarrow$ Estado actual ($\neq T$)
2: $a \leftarrow \pi(s)$
3: Ejecutar a, observar r y s'
4: $Q(s,a) \leftarrow Q(s,a) + \alpha\ [\ r + \gamma\ argmax_{a'}\ Q(s',a') - Q(s,a)]$
5: $Modelo\ (s,a) \leftarrow r, s'$
6: **Repetir** P veces:
7: $\quad s_p \leftarrow$ Estado previo
8: $\quad a_p \leftarrow$ Aleatoria en s_p
9: $\quad r, s'_p \leftarrow Modelo\ (s,a)$
10: $\quad Q(s_p,a_p) \leftarrow Q(s_p,a_p) + \alpha\ [\ r + \gamma\ argmax_{a'_p}\ Q(s'_p,a'_p) - Q(s_p,a_p)]$
11: $\quad s_p \leftarrow s'_p$
12: **Fin Repetir**
13: $s \leftarrow s'$

El Aprendizaje por Refuerzo consistiría en la selección de una acción dado el estado actual bajo una política (1,2), observando la respuesta del entorno (3) y actualizando el valor $Q(s,a)$ en función de lo anterior (4). El agente registra lo observado en su modelo interno (5), asumiendo un entorno determinista. La Planificación (6-12) distaría del Aprendizaje por Refuerzo en la toma de decisiones de forma aleatoria (8) y la observación de la respuesta del modelo y no del entorno (9), por lo que el agente no interactuaría de forma alguna con el entorno, siendo la experiencia que se genera simulada. Ésta interacción simulada se repetiría un número de pasos P. Los estados y acciones que se toman de forma simulada se han caracterizado con el subíndice p. Finalmente, tanto en Aprendizaje por Refuerzo y Planificación, se asignaría como estado actual el estado siguiente (11,13), continuándose así el proceso. El final estará marcado por un estado terminal T, que representará el objetivo que busca el agente, y lo ejecutado hasta entonces se denominará como episodio. El aprendizaje puede tener lugar en un determinado número de episodios, aunque el carácter de este algoritmo le permite ser aplicado en tareas continuas (sin T, al ser el horizonte temporal infinito).

Cabe destacar que el valor $Q(s, a)$ es un estimador de $q(s, a)$, y su actualización es la que se realiza con el *Q-learning*. Se trata de un método de Diferencia Temporal 0 [3], al considerar sólo en su formulación la toma de decisión en el estado siguiente. La expresión surge de la ecuación obtenida en la expresión (1), trasladada a un proceso iterativo de actualización, y su punto fuerte recae en que en el estado siguiente considera el valor $Q(s', a')$ máximo, buscando así directamente la consecución de acciones que permite maximizar la recompensa. Se ha demostrado la convergencia de este procedimiento a $q(s, a)$ con probabilidad 1 [4], independientemente de la política seguida por el agente. Parámetros destacables de la expresión serían el ratio de aprendizaje α, que toma valores entre 0 y 1 y marca en que medida el nuevo valor corrige al anterior en la actualización; y el factor de descuento γ, que cobra el mismo significado que el explicado anteriormente.

3. Implementación

La implementación del *Dyna-Q* impone una definición más concreta del problema de navegación dentro de un mapa que se trata en este estudio. En primer lugar, se va a determinar como un proceso discreto, dándose en intervalos de tiempo iguales y consecutivos. El mapa corresponderá al espacio de estados S del agente, definiendo en él un número de regiones rectangulares con las mismas dimensiones que conformarán los estados s. La dinámica del agente quedará limitada al movimiento en las cuatro direcciones del espacio (N-S-E-O), pasando así a los estados adyacentes en cada una de ellas (siempre y cuando sea posible). A su vez, la dinámica del entorno también quedará limitada y será inequívoca, de manera que las mismas acciones en un mismo estado conducirán al mismo estado siguiente y las mismas recompensas, por lo que nos situamos en un caso determinista. El modelo del agente incluirá el espacio de estados y su adyacencia, así como el sistema de recompensas y los estados inicial y final. Aquellos estados ocupados total o parcialmente por obstáculos serán estados no accesibles para el agente, y serán desconocidos por éste hasta que sean detectados. La capacidad de detección simulará un sistema de sensores implementados en el agente que le aportarán información exacta sobre la existencia de obstáculos en los estados circundantes al que se encuentre.

Este problema se ha planteado desde la perspectiva de que el proceso de aprendizaje debe darse en un sólo episodio. Con este fin se pretende asemejarlo a una situación real, con un agente móvil que deba resolver la ruta dentro del mapa conforme la ejecuta, careciendo de un número de repeticiones sobre el mismo mapa y bajo las mismas condiciones.

3.1. Diseño

El algoritmo *Dyna-Q* supone un punto de partida para el que finalmente se utilizará en la resolución del problema. Éste presenta unos parámetros que será necesario definir, para obtener una solución que se adapte a la naturaleza del problema. El ajuste de éstos se realizará o bien siguiendo definiciones de los mismos expuestas en la resolución de otros problemas similares o realizando un ajuste preciso para el problema que se desea resolver en particular, o bien mediante la evaluación de sus repercusiones sobre la solución obtenida. A continuación se expondrán las definiciones de los parámetros para el caso del estudio y/o la necesidad de evaluar su influencia sobre la solución.

Política Aunque la convergencia del algoritmo está asegurada, la política debe representar el comportamiento inmediato del agente tal que sea capaz de ejecutar la política óptima con arreglo a unos criterios. Se ha elegido la política denominada como ε-*greedy*, al combinar la exploración con la elección de acciones óptimas. Se define así un parámetro ε, de manera que con probabilidad $1 - \varepsilon$ el agente decida tomar la acción que le acerque hacia su objetivo, es decir, aquella que conlleva un valor de $Q(s, a)$ mayor, y con probabilidad ε decida explorar tomando una acción de forma aleatoria en el estado en el que se encuentre. A nivel de implementación, esta probabilidad se formula como un parámetro ξ aleatorio que toma valores de 0 a 1, de manera que si ε es mayor que ξ el agente explorará. La expresión (2) refleja de forma esquemática este método.

$$\pi(s) \leftarrow \begin{cases} \text{Aleatoriamente } a \in \mathcal{A}(s) & \text{si } \xi < \varepsilon \\ argmax_{\,a \in \mathcal{A}(s)}\, Q(s, a) & \text{si no} \end{cases} \tag{2}$$

La elección de ε se puede considerar como un parámetro más dentro del diseño del algoritmo. Una definición clásica es el inverso del número de episodios, de manera que en los primeros episodios se da una exploración exhaustiva, y conforme aumenta el número de episodios el agente toma las decisiones óptimas teniendo ya un mayor conocimiento del entorno. En el problema que concierne a este estudio en particular, al definirse la tarea en un solo episodio, ε dependerá del estado s en el que se encuentre el agente siendo el inverso del número de visitas a dicho estado. Esto implica que se exploren sólo aquellos estados en los que no se conozca el efecto de la toma de una determinada acción al haber recibido un menor número de visitas. En primera instancia, se considerarán de igual forma las visitas en experiencia directa y simulada, pero se profundizará en este concepto más adelante y se evaluarán sus repercusiones en cuanto a la ejecución del agente.

Ratio de aprendizaje Un entorno determinista influye sobre los valores $Q(s, a)$, siendo siempre la actualización de éstos correcta en términos de cálculo con respecto a lo experimentado en la interacción, ya sea directa con el entorno o a través del modelo. Este motivo justifica una elección del parámetro α igual a 1, ya que los valores $Q(s, a)$ no necesitan converger a un valor esperado que represente la distribución de los diferentes efectos de s y a, como ocurre en los entornos estocásticos.

Sistema de recompensas El sistema de recompensas que se ha establecido busca orientar al agente hacia el objetivo de efectuar la ruta óptima: aquella que partiendo de un estado inicial alcance otro final en el menor número de movimientos y sorteando los obstáculos que encuentre. De esta forma, llegar al destino conlleva la máxima recompensa r_{goal}, e intentar acceder a un estado con un obstáculo tendrá un recompensa negativa r_{obj} y con un valor igual al de r_{goal}. Realizar el menor número de movimientos se expresará mediante una recompensa por transición (movimiento) de un estado al siguiente r_{trans}, que será negativa y tomará un valor de acuerdo a un ratio k_r, definido como el cociente entre r_{goal} y r_{trans}. Teniendo en cuenta que el agente busca aquellas acciones que maximizan los valores de $Q(s, a)$, si se "penaliza" el realizar movimientos mediante una recompensa negativa, el agente evitará en la mayor medida posible el realizar movimientos innecesarios y, por tanto, buscará la trayectoria que recorra el menor número de estados. El aspecto relativo a las recompensas se retomará y discutirá más adelante dentro de este estudio.

Finalmente, el factor de descuento γ y el número de pasos de Planificación P también serán objeto de un posterior análisis y evaluación de su influencia sobre el comportamiento del agente.

3.2. Extensión al caso Multiagente

Uno de los aspectos relevantes de este estudio es la aplicación del algoritmo de aprendizaje *Dyna-Q* al caso multiagente, de manera que sobre un mismo entorno varios agentes sean capaces de alcanzar un serie de destinos. La aplicación del Aprendizaje por Refuerzo en el caso multiagente se ha tratado desde varias perspectivas, e incluso se ha planteado un método denominado *M-Dyna-Q* [5] que enfoca la toma de decisiones de forma conjunta entre los agentes. El método que se va a exponer en esta sección mantiene por separado la toma de decisiones para cada uno de los agentes.

Siguiendo la premisa expuesta para el caso de un agente, los agentes conocerán en qué estado se encuentra cada uno de ellos (sólo el propio) al iniciar el problema y cuales son todos los estados finales alcanzables. Por tanto, la solución óptima será aquella en la que los agentes tracen trayectorias hacia el destino más cercano, tomando dicha decisión de forma individual, si bien es cierto que la aproximación que se ha planteado para este problema implicará una comunicación entre los agentes (bien mediante una unidad central externa o bien directamente entre ellos) cuando uno de ellos llegue a uno de los destinos, en cuyo caso permanecerán ahí hasta que ningún destino quede sin alcanzar. Esta premisa se ha tomado de nuevo en vistas a una implementación en un sistema de agentes físicos, donde la comunicación también es un factor determinante y complejo. Otro enfoque sería que al comienzo de la tarea los agentes se comunicaran entre sí para conocer la posición del otro y asignar los destinos correspondientes, lo cual implicaría una transmisión de una gran cantidad de datos, así como un proceso de evaluación de las trayectorias, todo ello suponiendo que las trayectorias son directas entre inicio y final al no conocer los agentes los obstáculos del mapa y cómo afectarán en su alcance a los destinos, haciéndose necesarias evaluaciones de las trayectorias conforme de desarrollan. Mediante el método que se va a exponer, éste último problema también se va a solventar, al considerar sólo los agentes en sus decisiones aquellos destinos no alcanzados en un cierto instante.

El planteamiento que se ha tomado se basa fundamentalmente en conseguir que los agentes siempre orienten sus decisiones hacia el objetivo más cercano. Si bien es cierto que se podrán incurrir en soluciones con trayectorias que se alejen de las óptimas, tal y como se ha barajado frente a otras aproximaciones, se ha considerado como eficaz. La toma de decisiones de los agentes siempre se basa en los valores $Q(s, a)$, tomando aquellas acciones que los maximicen y por tanto les conduzcan hacia el objetivo. En el caso multiagente al existir varios destinos alcanzables, cada uno de ellos deberá llevar asociado unos $Q(s, a)$, siendo diferentes entre destinos para unos mismos s y

a. De esta forma, el agente para tener en cuenta el efecto real de sus acciones con respecto al conjunto de destinos, deberá orientar su política buscando tomar la acción que maximice su recompensa sobre todos los destinos alcanzables (de acuerdo a la política ε-*greedy* ya definida). Para ello, en s se tomará aquella a tal que el sumatorio de los valores $Q(s,a)$ para los destinos disponibles (no alcanzados en ese instante) sea mayor que para el resto de acciones que pertenezcan al espacio de acciones $\mathcal{A}(s)$. Esto hará que el valor de los sumatorios para cada a refleje los efectos de todas ellas para todos los destinos, y por tanto que acciones conllevarán una mayor recompensa.

Algoritmo 2 Extensión planteada del *Dyna-Q* para el caso multiagente

Inicialización: Matrices $Q_i(s,a)$ para los i Destinos y $Modelo\ (s,a)$ para todo $s \in \mathcal{S}$ y $a \in \mathcal{A}(s)$.

Hasta estado terminal T:

$s \leftarrow$ Estado actual ($\neq T$)
$a \leftarrow \varepsilon$-*greedy* : $\sum^i Q_i(s,a) |\ i$ no alcanzado ; $\forall\ a \in \mathcal{A}(s)$
Ejecutar a y observar s'
Para los i Destinos
  Si i no alcanzado **entonces**
    Observar r_i
    $Q_i(s,a) \leftarrow Q_i(s,a) + \alpha\ [\ r_i + \gamma\ argmax_{a'}\ Q_i(s',a') - Q_i(s,a)]$
  Fin Si
Fin Para
$Modelo\ (s,a) \leftarrow r, s'$
Repetir P veces:
  $s_p \leftarrow$ Estado previo
  $a_p \leftarrow$ Aleatoria en s_p
  $s'_p \leftarrow Modelo\ (s,a)$
  Para los i Destinos
    Si i no alcanzado **entonces**
      $r_i \leftarrow Modelo\ (s,a)$
      $Q_i(s_p,a_p) \leftarrow Q_i(s_p,a_p) + \alpha\ [\ r_i + \gamma\ argmax_{a'_p}\ Q_i(s'_p,a'_p) - Q_i(s_p,a_p)]$
    Fin Si
  Fin Para
  $s_p \leftarrow s'_p$
Fin Repetir
$s \leftarrow s'$

Todo lo expuesto implica que se deban observar las recompensas asociadas a la ejecución de una acción y la actualización de los valores de $Q(s, a)$ para todos los destinos que el agente deba considerar, tanto en la parte de Aprendizaje de Refuerzo como en la de Planificación. El Algoritmo 2 muestra de forma esquemática como se extendería *Dyna-Q* para los agentes de acuerdo a este planteamiento.

Este planteamiento también conduce a una caracterización del mapa en el que se desarrolle el problema, a la hora de clasificarlo en función del número de movimientos mínimo entre estados inicial y final (de aquí en adelante, tamaño de trayectoria). Si bien es cierto que el tamaño de trayectoria en los casos de un agente toma una definición clara, en el caso multiagente, cada uno de ellos deberá seguir una trayectoria diferente, y que será óptima si es hacia el destino más cercano. Por este hecho, aquella trayectoria que tenga un mayor número de movimientos será la que marcará el final de la ejecución, que se dará por finalizada cuando todos los estados sean alcanzados. Así se van a caracterizar los problemas multiagente por la trayectoria con el mayor tamaño. Esta cuestión será tratada y se evaluará si es adecuada más adelante dentro de este estudio.

Finalmente cabe destacar que siempre se va a considerar el mismo número de destinos y agentes. El planteamiento tomado no cambiaría y presentaría en los agentes el mismo comportamiento tanto si hay más destinos que agentes o viceversa, pero tal y como se ha definido el problema desde el inicio como trayectorias con un sólo destino final, quedarían destinos sin alcanzar o agentes sin destino.

3.3. *Robotic Motion Toolbox*

Como marco de trabajo se ha utilizado la plataforma *Matlab*, y dentro de ésta, la librería *Robotic Motion Toolbox* [6]. Ésta *toolbox* permite la planificación de trayectorias de uno o varios agentes móviles (sin que exista ningún tipo de proceso de aprendizaje por su parte en el que tomen decisiones sobre sus movimientos), así como la determinación de las señales necesarias para su control, según su morfología. Los parámetros, así como el mapa con el que se desea trabajar pueden ser dispuestos por el usuario, mediante la interfaz gráfica que se muestra en la Figura 3, que también permite la visualización del proceso. Con la implementación del algoritmo *Dyna-Q* y otras funciones relacionadas con la emulación del sistema de detección del agente y la generación de entornos, se ha contribuido a su mejora mediante la inclusión de funcionalidades relacionadas con procesos de aprendizaje. Algunas de las principales funciones que se han desarrollado se incluyen en los Apéndices A, B y C.

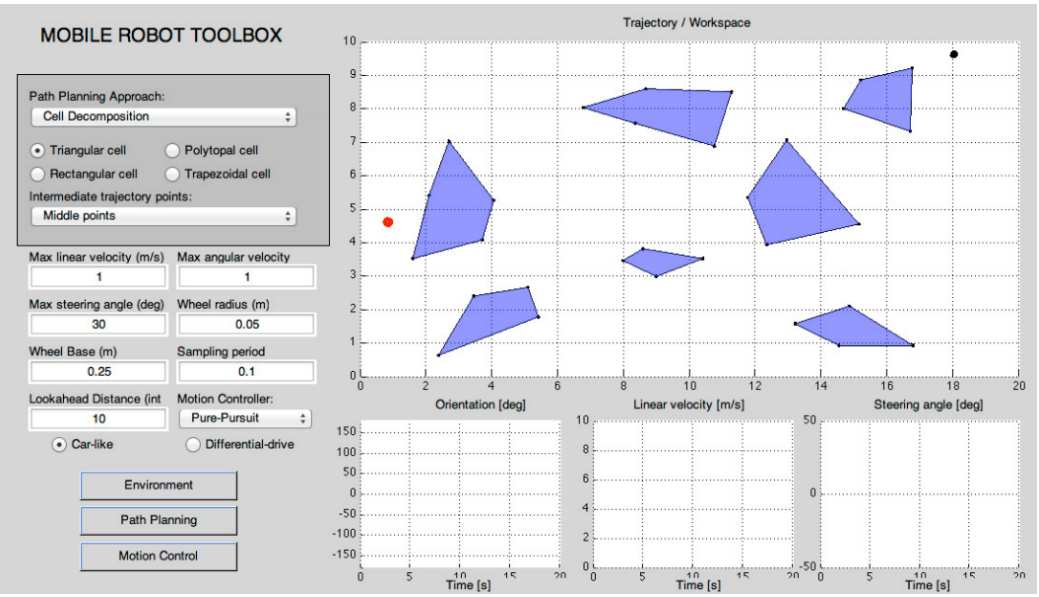


Figura 3: Interfaz gráfica de la librería *Robotic Motion Toolbox*

4. Simulaciones

Como se ha expuesto anteriormente, el algoritmo final que se utiliza en la resolución del problema está caracterizado por el valor o las definiciones de sus parámetros, al determinar en gran medida el nivel de adaptación de la solución al problema dado. En la sección de diseño del algoritmo, a algunos de estos parámetros se les ha dado una definición concreta, basándose en métodos conocidos o partiendo de éstos para generar una definición adaptada al problema. Sin embargo, algunas variables de estas últimas definiciones y otros parámetros no se han determinado de forma precisa, al desconocer cómo su elección influirá sobre la ejecución final. Por este motivo, es necesario realizar una evaluación objetiva de éstos analizando la solución obtenida bajo diferentes definiciones y/o valores. De esta forma, en esta parte del estudio se planteará para cada aspecto o parámetro una serie de simulaciones que, tras la interpretación de sus resultados, permitirán formular conclusiones acerca de su influencia y/o evaluar hipótesis o premisas que se planteen sobre ellos.

Esta parte de experimentación y simulación cobra gran importancia en la adaptación y definición de los algoritmos de aprendizaje, más aún cuando el número de parámetros a evaluar es notable. Por norma general en estos últimos casos, las experimentaciones se realizan de forma individual para cada parámetro, aislando las conclusiones obtenidas con cada experimentación del resto. Aunque este planteamiento no es erróneo, puede omitir el hecho de que existan interrelaciones entre distintos parámetros que determinen la solución, lo cual es entendible si se parte de la premisa de que la experimentación tiene como objetivo evaluar la influencia de los parámetros de los que se desconoce el comportamiento o se desea evaluar para el problema en concreto. En las experimentaciones que conciernen a este estudio se pretende seguir esta última idea, intentando cohesionar los resultados obtenidos con un parámetro en la evaluación del resto. La experimentación "idónea" para ello sería aquella que evaluara todas las combinaciones posibles de las definiciones y/o valores de los diferentes parámetros, pero debido a la exhaustiva computación que requeriría se ha optado por la propagación de los resultados obtenidos en algunas simulaciones hacia el resto. Esto se hará de acuerdo a un criterio de importancia y relevancia de los parámetros, y se utilizarán los valores que hagan que la solución del problema presente una mejora respecto a la solución inicial. Dicha mejora se valorará con arreglo a que el resultado sea mejor en cuanto a un menor número de movimientos, a un objetivo que se plantee en concreto, o a que la solución se obtenga con una utilización menor de los recursos de cálculo y/o memoria por parte del algoritmo. Este último criterio se enmarca de nuevo en los casos de aplicación a un sistema real,

donde los agentes presenten ciertas limitaciones físicas que obstaculicen la implementación del algoritmo.

En el caso particular de este estudio, la evaluación de los parámetros se hace especialmente relevante en el algoritmo *Dyna-Q*. El proceso de Planificación orienta en gran medida la trayectoria del agente generando un gran número de actualizaciones de los valores $Q(s, a)$, en torno a los cuales el agente orienta su toma de decisiones. Sin embargo, dentro de un estado la acción para la cual se actualiza su valor es puramente aleatoria. Esto conduce a que el comportamiento del agente es en fundamento aleatorio, y la elección de unos parámetros adecuados permitirá también salvar esta aleatoriedad y asegurar en la mayor medida posible comportamientos que escapen a los deseados. De esta forma, las simulaciones se deberán plantear para que recojan el comportamiento del agente un determinado número de veces tal que se resuma el comportamiento que presenta el agente bajo unos mismos parámetros.

Por otro lado, la experimentación también tendrá como objetivo verificar que el método empleado para la extensión del problema al caso multiagente es viable y eficaz. Debido a la complejidad que se presenta a la hora de caracterizar de forma precisa y sencilla un caso multiagente, se complica la realización de una experimentación que arroje resultados concluyentes. De esta forma, con cada parámetro se realizará la experimentación para el caso de un agente, y los resultados que presenten una mejor solución del problema se aplicarán a simulaciones sobre casos multiagente. La obtención de soluciones semejantes se entenderá como la aceptación de la viabilidad de esta aproximación.

4.1. Metodología

En lo que se refiere a las experimentaciones, se ha tomado como condición general que los parámetros y variables se evalúen en un rango amplio donde se suponga a priori que se van a observar diferencias en la ejecución de las soluciones. Éstas se van a caracterizar principalmente por el número de movimientos del agente en su trayectoria entre el punto inicial y el final. Dentro de ese rango, se deberán obtener soluciones para un número de valores suficiente tal que entre valores consecutivos no exista una variación no previsible en los resultados y que, por tanto, no sea observable en ellos.

Estas consideraciones determinan unas experimentaciones muy exhaustivas, en lo que respecta a computación. Para solventar esto, se barajaron posibilidades que permitieran trabajar utilizando el entorno de *Matlab*, en la que se han implementado todas las funciones y algoritmos necesarios. La propia plataforma ofrecía opciones de ejecución en paralelo de procesos (*Matlab Parallel Pool*), limitadas por las características de procesamiento del equipo. Tras comprobar que incluso con esta opción las experimentaciones tenían tiempos de ejecución de varios días, se optó por el clúster de computación de alto rendimiento HERMES, desarrollado en el Instituto de Investigación en Ingeniería de Aragón (I3A). Este clúster cuenta con el sistema de gestión Condor, que permite una ejecución simultánea de tareas. De esta forma, todas experimentaciones se han diseñado con arreglo a este sistema, o bien para consumir un menor tiempo de ejecución, o bien para dentro del mismo tiempo de ejecución la realización de múltiples experimentaciones.

Los parámetros y variables expuestos en la sección de diseño del algoritmo como objeto de análisis y evaluación serán los sometidos a las experimentaciones. Se han dividido en tres bloques representativos, según afecte sobre un aspecto en concreto del comportamiento del agente: planificación, cálculo y exploración. Las simulaciones que se desarrollen en estos bloques partirán de unos valores para dichos parámetros que se ha observado que aseguran una solución en un número de movimientos cercano al mínimo para cualquier configuración de mapa. El valor de los parámetros se cambiará en función de la experimentación, ya sea porque es el que se evalúa en dicha simulación en cuestión o porque se fija un valor concreto en función de otras simulaciones. Dichos valores los denominaremos como estándar y serán los que se muestran en la Tabla 1.

Parámetro	P	r_{goal}	k_r	γ
Valor	15000	10^7	1000	0,9

Tabla 1: Valores estándar de los parámetros

4.1.1. Planificación

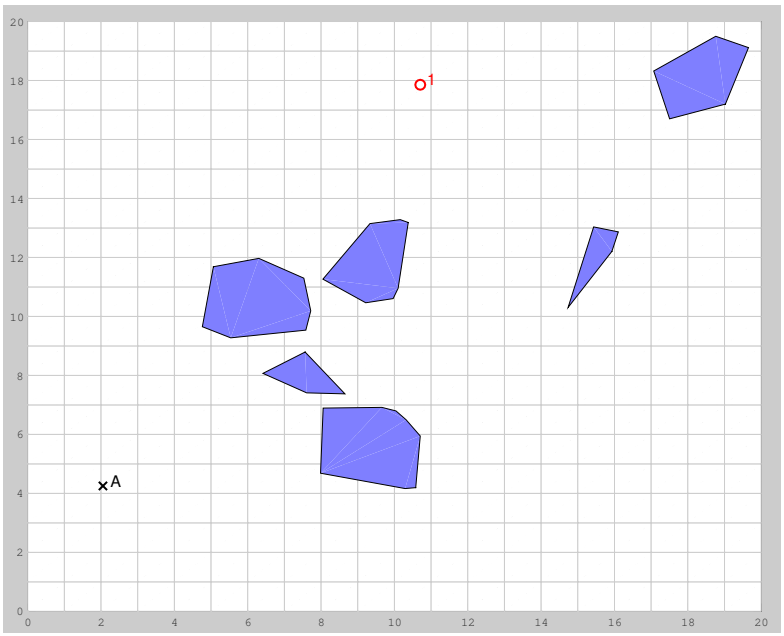
Dentro de este bloque se encontrarían aquellos que afectan sobre la fase del algoritmo *Dyna-Q* con el mismo nombre. El parámetro que influye de forma más directa es el número de pasos de planificación P. La determinación de este parámetro es crítico en el proceso de aprendizaje, ya que limita el número de movimientos que realiza con experiencia simulada el agente por cada movimiento que hace sobre el entorno real. Esto afecta directamente

sobre su capacidad para establecer la consecución de movimientos que le conducirán al destino, ya que a menor P, previsiblemente, se explorará un menor número de estados y por tanto se actualizará un menor número de valores $Q(s, a)$. Si no se actualizan, no tomarán el valor real de ejecutar una acción a dentro de un estado s, y en el caso que el agente alcance el estado s buscando maximizar la recompensa, maximizará sobre unos valores erróneos que puede que le conduzcan a alejarse del destino. Este parámetro también debe de establecerse atendiendo al hecho de que el proceso de Planificación tiene una alta variabilidad, al ser aleatoria la toma de acciones en esta parte. Por ello, el valor que tomará P tenderá a ser elevado, ya que se tendrá que asegurar generar actualizaciones suficientes de estados como para que el agente pueda orientar sus acciones de forma correcta a lo largo de la trayectoria. La distancia entre el estado de inicio y final también determinará el valor de P, puesto que cuanto más cercano se encuentre el destino, a priori, será necesaria la actualización en un número menor de estados. También cabe destacar que este parámetro es uno de los que cobra más importancia en cuanto a la aplicación del algoritmo sobre agentes físicos, al afectar sobre el procesamiento necesario para realizar la parte de Planificación.

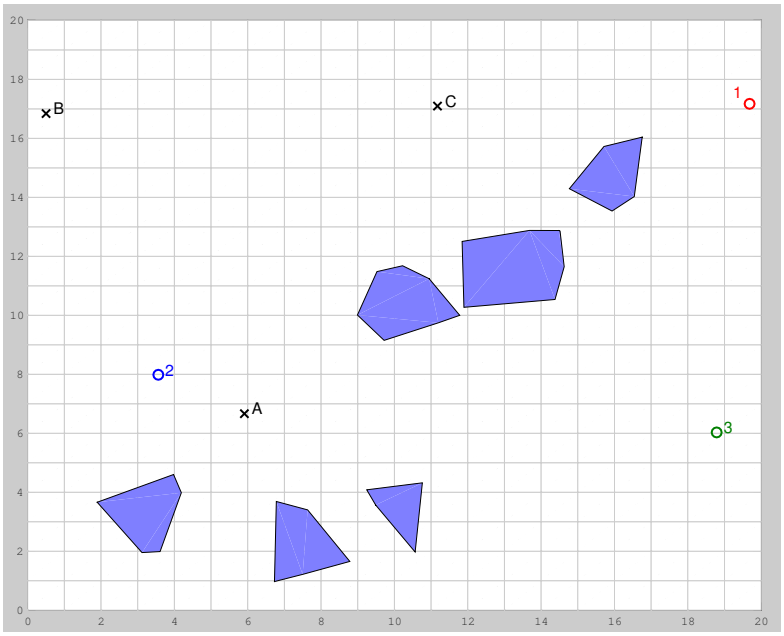
La experimentación en el caso de un agente se realizará sobre mapas formados por 400 estados (20×20), generados de forma aleatoria con el mismo número de obstáculos y tamaño máximo (dentro de este tamaño, su contorno será aleatorio). Se evaluará el parámetro P en un rango de 10 a 15000, y en mapas con un tamaño de trayectoria entre 5 y 35, en intervalos de 5. Así, se ha realizado la experimentación para cada tamaño de trayectoria en 5 mapas distintos, dentro de los cuales cada valor de P se ha evaluado 10 veces. En la Figura 4 se muestran ejemplos de entornos generados para ambos casos. Se considerará como solución óptima aquella que con un menor valor de P asegure en cada caso una ejecución con un error relativo del 25 % sobre el tamaño de trayectoria.

El caso multiagente se experimentará para problemas con 2 y 3 agentes, considerándose siempre el número de destinos y agentes el mismo. Se aplicarán los valores de P que se consideren como solución óptima para cada tamaño de trayectoria, que tal y como se ha mencionado en la sección de extensión al caso multiagente 3.2, será el tamaño de trayectoria mayor. Los valores se aplicarán para cada tamaño de mapa sobre 50 distintos (generados bajo las mismas características que en el caso de un agente), y dentro de cada uno de ellos 10 veces. El número de mapas se justifica por ser la variabilidad del problema mayor al poder existir infinitas combinaciones de posiciones de agentes y destinos para un mismo tamaño de trayectoria, además de que al comprobar un valor determinado no se requiere tanto consumo de tiempo y

recursos como cuando se realiza un barrido en un rango de valores.



(a) Para 1 agente



(b) Para 3 agentes

Figura 4: Ejemplos de mapas generados aleatoriamente para las experimentaciones de P

4.1.2. Cálculo

Dentro de este bloque se incluirán los parámetros que actúen sobre la actualización de los valores de $Q(s, a)$, tanto en la parte de Aprendizaje por Refuerzo como en la de Planificación. Este bloque no se considera tan fundamental como el de Planificación, ya que depende en gran medida de él al marcar el ritmo de las actualizaciones en el proceso de aprendizaje y no sería tan crítico en cuanto a obstaculizar la aplicación sobre agentes físicos, ya que afecta en parte a los valores numéricos que toman $Q(s, a)$ y se almacenarían en memoria. Los parámetros incluidos en este apartado serán r_{goal}, el ratio k_r que determinará el valor de r_{trans}, y el factor de descuento γ. Ambos están presentes de forma implícita en la formulación de la actualización procedente del *Q-learning*. En primer lugar, r_{goal} será el que marque la escala numérica de los valores $Q(s, a)$, al ser el que determina el valor del resto de recompensas. El parámetro k_r establecerá el valor de r_{trans} necesario tal que los valores de $Q(s, a)$ reflejen la trayectoria que recorra un menor número de estados. El factor de descuento γ marcará dentro de la actualización el peso de tomar en el próximo estado la acción que maximice el valor de $Q(s, a)$ y, por tanto, le conduzca hacia el destino. Siguiendo esta definición, este parámetro determinará la propagación de los valores de $Q(s, a)$ hacia sus estados predecesores, por lo que situándonos en el estado de destino, γ determinará el número de estados en los que existan un valores de $Q(s, a)$ que reflejen qué acción conduce hacia el destino. De esta forma, valores mayores de γ supondrán que los valores en $Q(s, a)$ reflejen en mayor medida maximizar la acción en los estados posteriores y, por tanto, en estados alejados del destino se mantendrá esa tendencia.

En cuanto a la experimentación, se ha realizado únicamente sobre γ, fijando los valores estándar para r_{goal} y k_r. Se ha considerado que una evaluación sobre γ será de mayor interés, al reflejar el concepto de 'alcance' de un destino sobre el resto de estados, lo cual podría ser determinante en los casos multiagente, al basarse la toma de decisiones tal y como se ha planteado en la superposición de los valores $Q(s, a)$ para cada uno de los destinos. De esta manera, se evaluará la trayectoria del agente sobre mapas con las mismas características que los planteados en el caso de un agente para la experimentación con P, y γ tomará valores en un rango de 0,01 a 0,99. Tanto en el caso de un agente como en el de múltiples se realizará la experimentación con el mismo número de mapas e iteraciones por cada valor que en el caso de P, y sobre los mismos tamaños de trayectoria. En el caso de un agente, el menor valor de γ asegure un 25 % de error relativo sobre el tamaño de trayectoria se considerará como solución óptima, y sus resultados se evaluarán en los casos

de 2 y 3 agentes. Cabe destacar que en todas estas experimentaciones se han fijado los resultados de P óptimos, tal y como se ha justificado al comienzo de esta sección.

4.1.3. Exploración

Comprende los parámetros que afectan sobre el comportamiento del agente con respecto a la exploración, y cómo ésta determina el conocimiento que adquiere sobre el entorno. Su importancia recae en que una mayor interacción con el mapa en el que se desarrolla el problema le permitirá al agente corregir el modelo inicial (donde no se incluye ningún obstáculo), y así la fase de Planificación se podrá llevar a cabo de acuerdo a un modelo más exacto. Tal y como se ha definido la política *ε-greedy* en este estudio, la toma de una acción con el fin de explorar es inversamente proporcional al número de visitas que recibe tanto en interacción real como simulada. Con respecto a esta definición se van a introducir tres nuevos parámetros, que serán objeto de la experimentación en este apartado. El primero será m, y consistirá en un factor que multiplicará a ε, y será constante a lo largo de todas las ejecuciones. Presumiblemente, cuanto mayor sea m se necesitarán más visitas para que ε sea menor que ξ, y por tanto el agente elegirá la exploración un mayor número de veces. El segundo y tercer parámetro se denotarán por k_v y n_v respectivamente, y se definirán en torno a los valores que se contabilizan como visitas para el cálculo de ε. Por un lado, k_v se formulará como el cociente entre el valor de la visita en la interacción directa con el entorno y en la experiencia simulada, y será constante a lo largo de la ejecución. El parámetro n_v formará parte de una función que definirá el cociente k_v como variable según el número de movimientos t realizados en la ejecución de la trayectoria, siguiendo k_v una tendencia creciente y asintótica a un valor v, conforme aumenta t. Esta función se representa mediante la expresión (3).

$$k_v = \frac{-n_v}{t} + v \tag{3}$$

La asignación de valores distintos a las visitas se puede justificar atendiendo a su naturaleza. Al visitar un estado dentro del entorno real y ejecutar una acción, el agente observa de forma directa el comportamiento del entorno y detecta los obstáculos circundantes, de manera que el resultado que obtenga de esa interacción será veraz. Pero cuando la interacción con el entorno se da a través de experiencia simulada en torno a un modelo, los resultados

dependerán de que éste represente fielmente al entorno. Por ello, si el agente parte de un modelo erróneo (sin obstáculos) tal y como se ha planteado el problema, valorar las visitas en experiencia simulada de la misma forma que las que se dan por interacción directa no resulta a priori una buena aproximación.

La experimentación del parámetro m se realizará en el caso de un agente en el mismo tipo de entornos que en los apartados de planificación y cálculo, y sobre los mismos tamaños de trayectoria, considerando de igual forma 5 mapas distintos para cada uno de ellos, y 10 repeticiones para cada valor del parámetro sobre un mismo mapa. Los valores que tomará m estarán comprendidos entre 1 y 10, y se considerará como solución óptima aquella que suponga una notable mejora sobre el 25 % de error relativo que se obtiene con los valores previamente elegidos para P (que también se fijan en las simulaciones de este apartado). Las soluciones obtenidas se evaluarán en los caso de 2 y 3 agentes, también en las mismas condiciones que en los apartados previos.

Con los parámetros k_v y n_v la experimentación se va a enfocar desde una perspectiva diferente a la tomada a lo largo de esta sección. En su experimentación se va a analizar la influencia de éstos parámetros sobre el conocimiento que el agente tiene sobre su entorno, recogido en su modelo interno. De esta forma, se evaluarán ambos por separado bajo un mismo entorno, que se muestra en la Figura 5. Este mapa se ha diseñado de tal forma que el conocimiento que tenga el agente del entorno sea muy relevante en una ejecución óptima. La evaluación en el caso multiagente en estas experimentaciones se realizará bajo la perspectiva de considerar la interacción entre agentes sobre el mismo mapa en tareas distintas, de manera que un primer agente se dedicará a una tarea plenamente exploratoria y el modelo más completo que se haya obtenido en un menor número de movimientos se transferirá al segundo agente, que previsiblemente con un mejor conocimiento del mapa será capaz de realizar su ejecución en un menor número de movimientos.

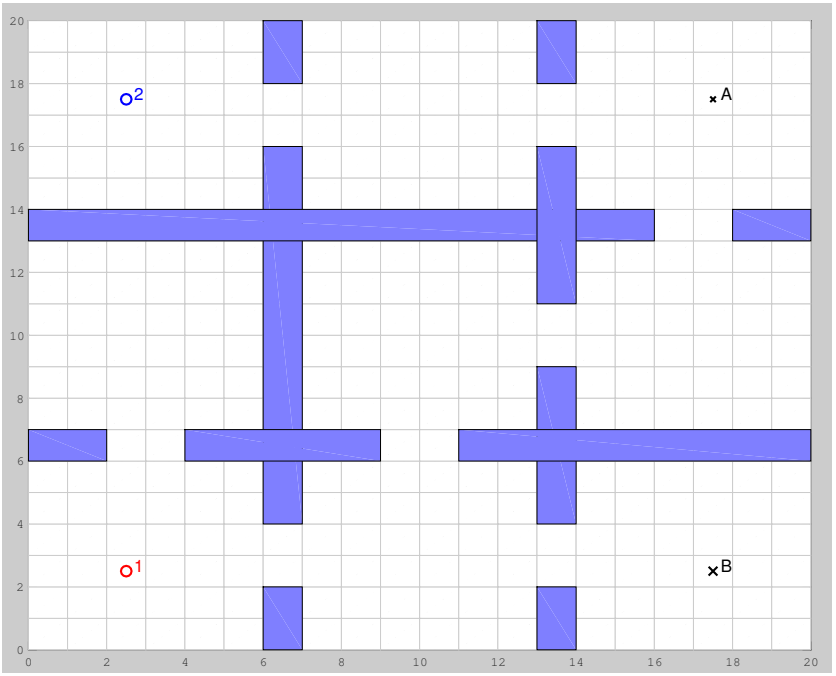


Figura 5: Mapa diseñado para las experimentaciones de k_v y n_v

De forma más concreta, en el caso de un agente el parámetro k_v ha tomado valores entre 1 (mismo valor para una visita en Aprendizaje por Refuerxo y en Planificación) y 50000, evaluándose sobre cada valor 50 veces. Para n_v se ha definido un rango de valores de 10 a 1000, evaluándose también cada valor 50 veces,y un v constante e igual a 50000. La solución óptima será aquella que con un menor número de movimientos en su trayectoria presente un mayor conocimiento del mapa, por lo que se evaluará el error relativo (%) sobre el tamaño de mapa y al completitud del modelo (%) sobre el total de estados ocupados por obstáculos. Los modelos obtenidos en esta solución serán los que se evalúen en la ejecución del segundo agente, de manera que se cada uno se evaluará 10 veces para luego obtener la media entre los todos los resultados y obtener el error relativo sobre el tamaño de trayectoria. Los estados iniciales y finales de cada agente serán distintos, y se muestran en la 5, siendo la trayectoria del primer agente definida entre 1 y A, con un tamaño de 30 movimientos, y la del segundo entre 2 y B, con uno de 42. Como se ha mencionado anteriormente, el valor del parámetro P también se fijará como el óptimo obtenido para cada una de las trayectorias, y en el caso de $2 - B$ se asignará el valor de 2500 correspondiente a 35.

5. Resultados

Tal y como se han planteado las simulaciones se han obtenido un cierto número de datos para unos mismos valores de parámetros con el fin de caracterizar el comportamiento del agente, debido a la aleatoriedad en la que se fundamenta el algoritmo *Dyna-Q*. De esta forma, en la representación de los resultados se va a considerar la media sobre todos los valores bajo unas mismas características, y el error relativo para evaluar la ejecución de las trayectorias. Pese a esto, el factor aleatorio del algoritmo siempre estará presente, por lo que se ha optado por el ajuste de dichas medias en el rango del parámetro que se experimente mediante un Proceso de Regresión Gaussiana (RGP) [7], que representará la tendencia de los datos y, por tanto, la del comportamiento del agente. La interpretación de los resultados de las soluciones óptimas se llevará a cabo respecto a esta tendencia teniendo siempre en cuenta que con esta consideración, las evaluaciones y soluciones se darán respecto a valores aproximados de los parámetros y no exactos.

5.1. Planificación

Los resultados obtenidos para el rango de valores de P para los distintos tamaños de trayectoria en caso de un agente se han representado respecto al error relativo en la Figura 6. El comportamiento del agente en la experimentación ha sido el esperado: un mayor número de P asegura una trayectoria en un menor número de movimientos, al disponer de una mayor cantidad de experiencia simulada por cada movimiento sobre el entorno. Con esta representación del error relativo se observa como para menores tamaños de mapa se necesita un mayor número de iteraciones para alcanzar el 25 % (representada en la Figura 6 como una línea gris punteada), lo cual se refleja de forma muy clara en el tamaño de trayectoria de 5 y de 35, que necesitan respectivamente valores de P de 15000 y de 2500. Esto se debe a que la medida del error relativo hace que se valore la diferencia absoluta entre el número de movimientos que el agente ejecuta y el tamaño de trayectoria (mínimo) sobre este último, por lo que en tamaños de trayectoria mayores una misma diferencia de movimientos conllevará un menor error relativo que en los de menor tamaño.

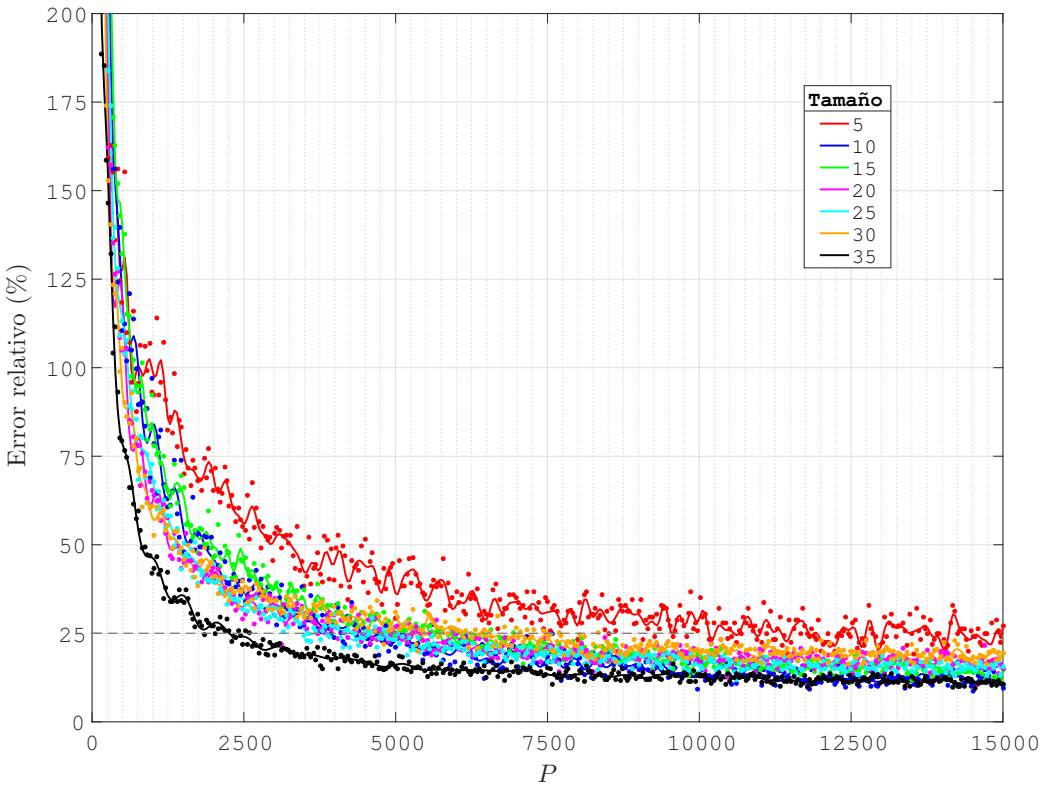


Figura 6: Error relativo (%) frente a P

A nivel aclaratorio, en la Figura 7 se ha querido representar esa diferencia absoluta, es decir, el número de movimientos que el agente sobrepasa el tamaño de la trayectoria. Se ha tomado un rango de P entre 0 y 2500, y en él se observa como los menores tamaños de trayectoria, con un valor muy bajo de P, consiguen que la diferencia absoluta (error absoluto) sea mucho menor que para mayores tamaños con el mismo valor.

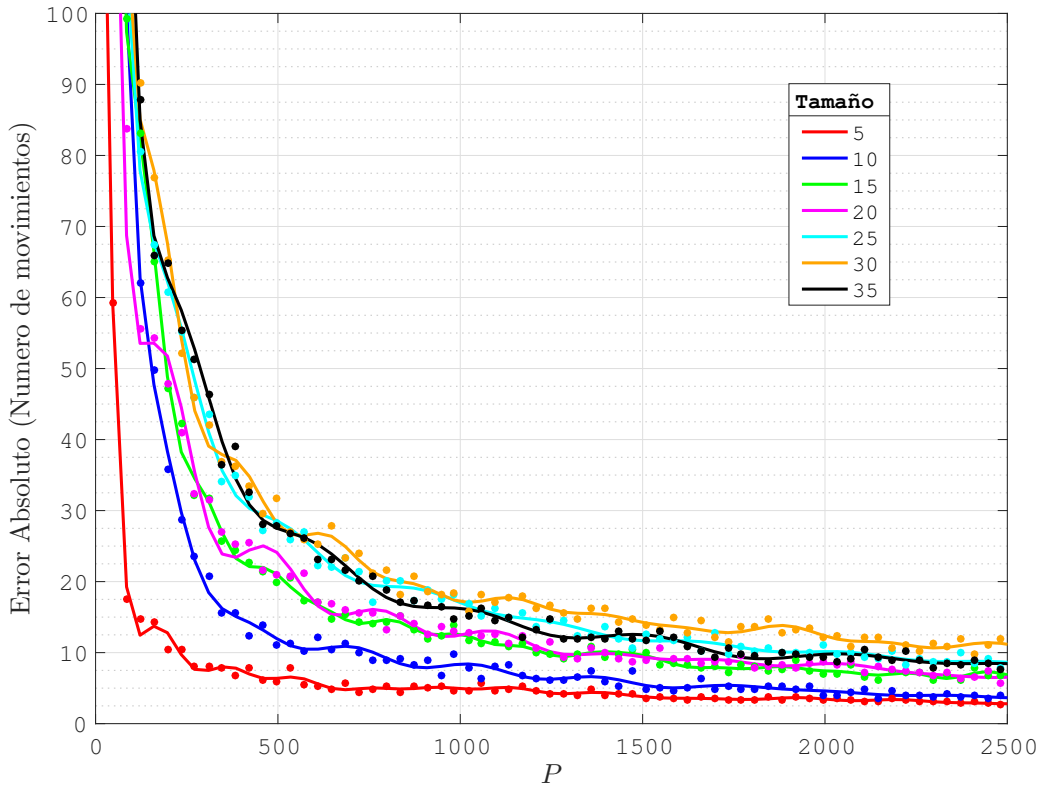


Figura 7: Error absoluto (Número de movimientos) frente a P

Las soluciones óptimas y que se fijarán como valores de P en los experimentos relativos al caso multiagente en esta parte y a todas las experimentaciones del resto de bloques, se ha establecido que serán aquellas con las que se alcance una ejecución del 25 % . Así para el tamaño de trayectoria de 5 el valor será de 15000, para los comprendidos entre los tamaños de 10 y 30 se observa un comportamiento similar, por lo que les ha asignado un valor común de 5500, y al tamaño de 35 se le asignará un valor de 2500.

Los resultados en términos de error relativo obtenidos para cada tamaño de trayectoria en los casos multiagente se muestran en la Tabla 2. En base a ellos se puede afirmar que la utilización de las conclusiones obtenidas para el caso de un agente en la extensión de *Dyna-Q* y la caracterización del problema que se ha planteado es factible para los mayores tamaños de trayectoria. Tal y como se observa, el error relativo se hace menor conforme mayor es el tamaño

de trayectoria, alcanzándose tanto en los casos de 2 como de 3 agentes para los tamaños de 25 a 35 errores relativos de alrededor del 25-30 %. En los tamaños de trayectoria menores a éstos los errores relativos, aunque en el caso de 2 agentes se mantienen por debajo de un error del 45 %, en el caso de 3 agentes la diferencia de errores se hace más notable.

	5	10	15	20	25	30	35
2 agentes	40.56	44.76	43.96	30.04	28.13	25.02	27.06
3 agentes	59.76	68.24	51.89	41.83	34	21.55	23.66

Tabla 2: Errores relativos (%) para los distintos tamaños de mapa y casos multiagente con los resultados de P

La diferencia de errores relativos entre los tamaños de trayectoria, así como aquella entre los casos de 2 y 3 agentes para un mismo tamaño, podrían interpretarse como un efecto derivado del uso del error relativo para evaluar la extensión al caso multiagente. Como se ha mencionado, el error relativo es una medida dependiente del tamaño de trayectoria sobre el que se evalúe, y el planteamiento tomado para el algoritmo en el caso multiagente hace que el comportamiento del agente se vea guiado hacia el objetivo más cercano. Si el agente con la trayectoria mayor ve afectado su comportamiento por otros destinos cercanos pero que serán alcanzados antes por otros agentes, presumiblemente se alejara de la trayectoria que debe de seguir y ello afectará a la ejecución total del problema, teniéndose un mayor error relativo para menores tamaños de trayectoria. En el caso de los 3 agentes es más probable que esta circunstancia se de al aumentar el número de destinos que puedan influir en su comportamiento. También cabe la posibilidad de que este efecto se deba a que la caracterización de los problemas mediante la trayectoria con el mayor número de pasos no sea útil a la hora de aplicar los resultados obtenidos en el caso de un agente, ya que en los entornos con 3 agentes la casuística posible aumenta.

5.2. Cálculo

Para el caso de un único agente se han representado los resultados obtenidos de las ejecuciones en forma de error relativo en función de los valores de γ en la Figura 8. En cuanto a la representación, para cada tamaño de mapa se ha observado que por debajo de un cierto valor de γ la ejecución se disparaba fuera de los rangos de error relativo esperado, creando una notable diferencia con aquellos que sí estaban dentro. Por ello se ha optado por representar sólo estos últimos y utilizarlos para ajustar el modelo RGP del tamaño de trayectoria correspondiente.

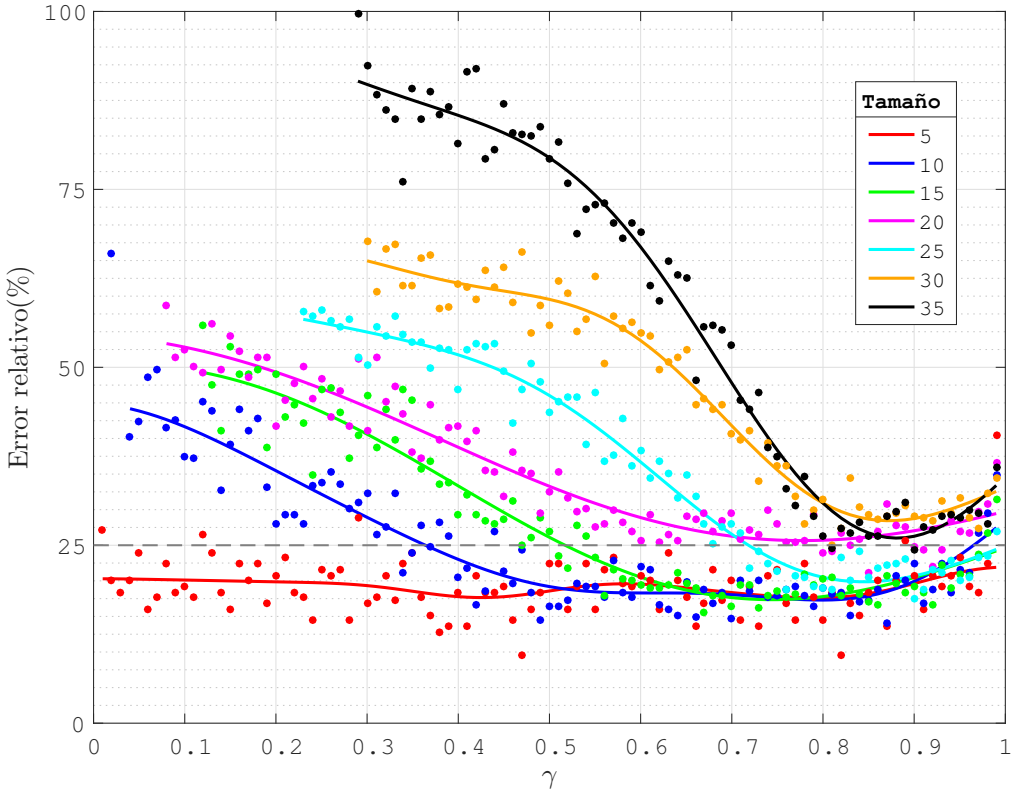


Figura 8: Error relativo (%) frente a γ

Tal y como se observa en la gráfica, mayores tamaños de mapa requieren que el valor de γ se acerque más a la unidad. Esto concuerda con las premisas que se habían supuesto sobre el comportamiento del agente que implicaba γ,

marcando su capacidad para tener en cuenta la búsqueda de la acción que maximice la recompensa en los estados futuros, por lo que en trayectorias mayores será necesario tener en cuenta un horizonte de estados futuros mayor.

Las soluciones óptimas de γ también se han considerado como aquellos valores que representen una ejecución con un error del 25 %. Para el tamaño de trayectoria de 5 se observa una ejecución óptima para todo el rango de valores, y esto puede deberse a que la consideración de fijar P como 15000 comporte en realidad unas ejecuciones mejores que las supuestas en dicho apartado. Por este motivo, para el tamaño 5 se asignará un γ mínimo, que se ha fijado en 0,1. Para los tamaños de 10, 15 y 25 se le asignarán los valores aproximados en los cuales su tendencia sobrepasa el error del 25 %, y serán 0,35, 0,55 y 0,75 respectivamente. Las funciones para los resultados de 20, 30 y 35 no llegan a alcanzar el 25 % así que se han tomado valores aproximados para los que alcanzan un mínimo, siendo 0,8 el valor para 20 y 0,9 para 30 y 35. Este fenómeno puede deberse a la existencia de alguna muestra errónea que desvíe de forma excesiva la media de los resultados.

	5	10	15	20	25	30	35
2 agentes	33.8	30.28	41.23	27.11	34.38	25.03	25.77
3 agentes	42.64	65.84	62.19	50.93	39.73	30.1	35.89

Tabla 3: Errores relativos (%) para los distintos tamaños de mapa y casos multiagente con los resultados de γ

Fijando éstos valores en el caso multiagente, se han obtenido los resultados que se muestran en la Tabla 3. Para el caso de 2 agentes se observa que el error relativo disminuye conforme aumenta el tamaño de trayectoria, tal y como en los resultados para P en el mismo caso, aunque respecto a este, en los tamaños de mapa menores se ha observado que el error relativo ha disminuido. Este hecho permite designar en este caso al parámetro γ como de ajuste fino de las soluciones obtenidas con P para obtener una más adaptada. Con 3 agentes los resultados el error relativo aumenta respecto a los resultados para 2 al igual que en mismo caso en P, pero ahora con respecto a este último no se ha obtenido ninguna mejora. Por lo tanto en este caso γ no se puede considerar como un parámetro de ajuste, y la diferencia con el caso de 2 agentes puede también derivarse de las causas de variabilidad de entornos y uso del error relativo expuestas también en el caso de 3 agentes en P.

5.3. Exploración

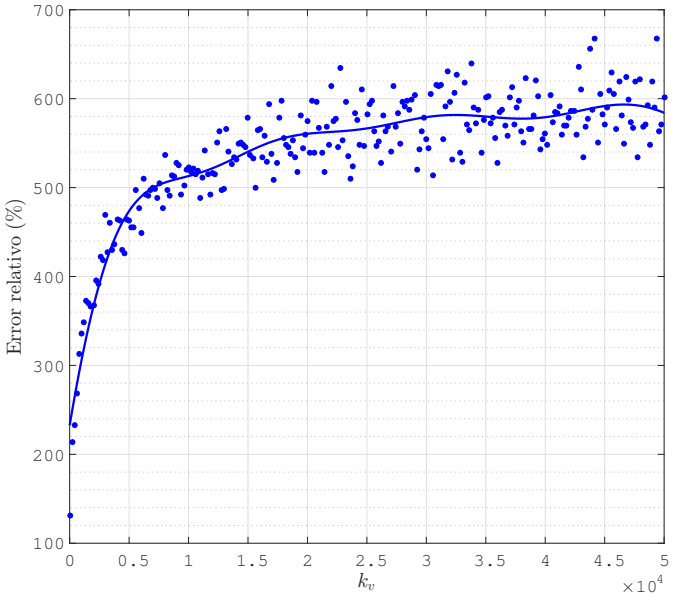
En primer lugar se ha planteado la experimentación del factor m. El efecto que se pretendía observar era si mayores valores de m implicaban una mejor ejecución de la trayectoria al aumentar la exploración del agente, partiendo del 25 % de error relativo que suponen los valores de P fijados. Los resultados observados concluyen que la exploración no reduce el número de movimientos en la trayectoria del agente, ya que sobre los tamaños de trayectorias sometidos en la experimentación el aumento del parámetro m sólo ha comportado un aumento del número de iteraciones, lo que implica que la exploración es innecesaria en el caso planteado. Puesto que no se ha obtenido ninguna solución óptima, no se evaluará el efecto de m en casos multiagente. En el Apéndice D se han representado algunos de los resultados de las experimentaciones como el error relativo en función de m.

Dentro de la parte referente a la evaluación de la exploración que el agente realiza sobre un mismo entorno, para el parámetro k_v se ha observado una diferencia notable sobre el conocimiento del entorno por parte del agente, mientras que para n_v no se ha observado ningún efecto apreciable. En ambos casos, se ha representado por un lado el error relativo y por otro la completitud (en %), ambos en función del parámetro correspondiente.

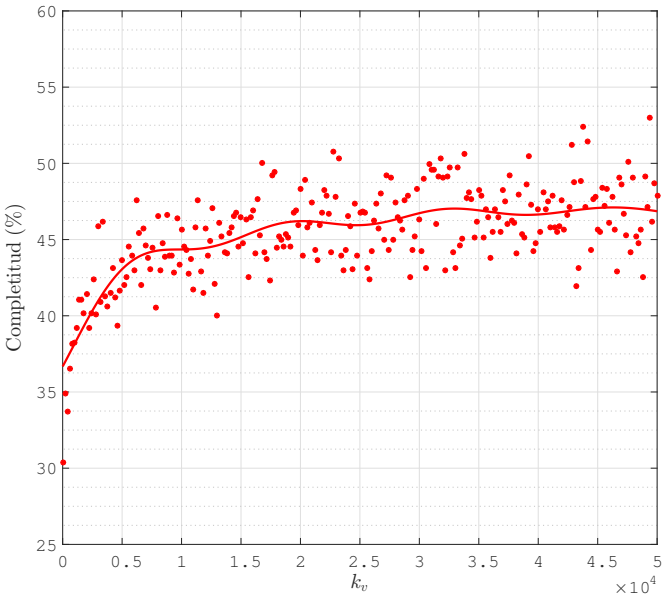
En la Figura 9 se muestran ambas representaciones para k_v, y reflejan un aumento brusco de la exploración y el número de movimientos de forma simultánea, hasta un punto en torno al valor de 30000 donde la completitud del mapa parece saturar por más que se aumente el número de movimientos, que pasa a partir de ese valor a crecer de una forma menos pronunciada. El comportamiento del agente tiene sentido, ya que cuanto menos contabilice la visita a un estado en la fase de Planificación (mayor k_v), según la política *ε-greedy* la toma de decisiones aleatoria a favor de explorar tendrá una mayor probabilidad de ocurrir. La saturación podría tener relación con el entorno en sí, ya que llegado cierto número de movimientos por parte del agente se aseguraría una exploración suficiente como para detectar los obstáculos que se encuentren dentro de la zona susceptible de exploración para la trayectoria definida entre sus estados inicial y final.

Por otro lado, el parámetro n_v no ha demostrado tener relación con la exploración del agente, presentando los mismos resultados en todo su rango. En comparación con lo obtenido para k_v se observa cómo los resultados de n_v son similares a los que presenta k_v en torno al valor de 50000, que es el que se había fijado como v en la expresión (3), por lo que se puede suponer que realmente es v el que ha determinado la experimentación y no n_v. En el

Apéndice E se han recogido las representaciones correspondientes a n_v.



(a) Error relativo (%) frente a k_v

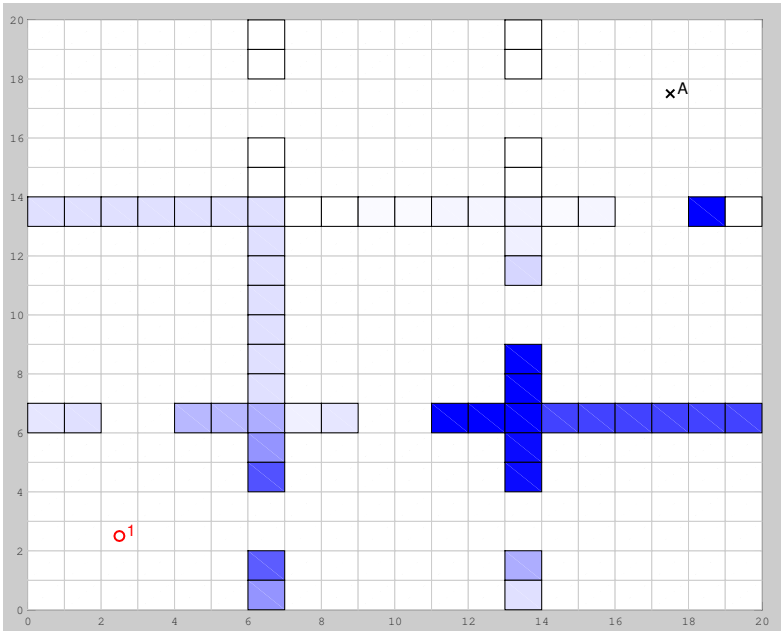


(b) Completitud del modelo(%) frente a k_v

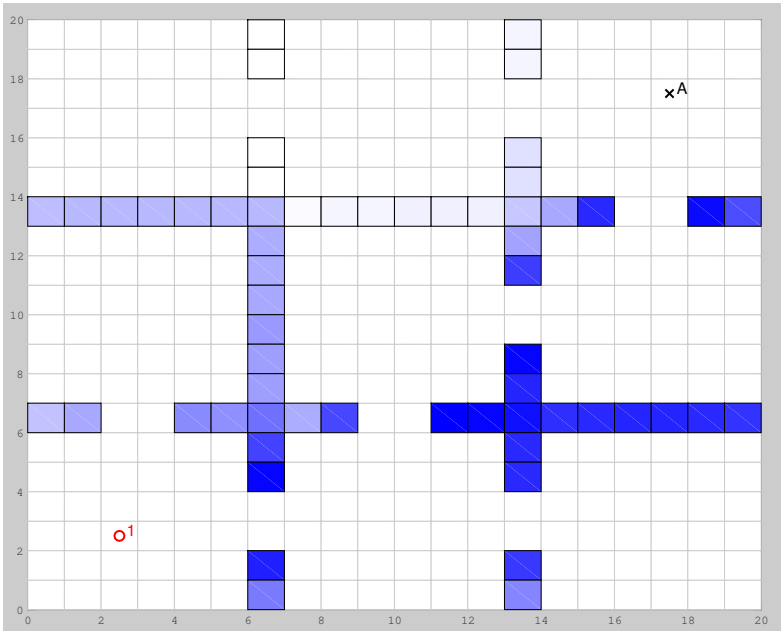
Figura 9: Completitud del modelo y error relativo en función de k_v

La experimentación para el caso multiagente, tal y como se ha expuesto, supone la evaluación de la ejecución de un agente al utilizar el modelo obtenido por un primer agente "explorador" anterior. Cómo modelos óptimos se han empleado los correspondientes al valor de k_v igual a 30000 mencionado anteriormente, ya que son los más completos obtenidos en un menor número de movimientos del agente. A modo de referencia se ha realizado la misma evaluación para un valor de k_v igual a 1, que correspondería a la situación de partida en la que el valor con el que se contabiliza una visita es el mismo para Planificación y Aprendizaje por Refuerzo. En primer lugar, para ilustrar el nivel de completitud del modelo sobre el mapa se ha optado por mostrar el número de veces que los obstáculos han sido detectados sobre el conjunto de modelos obtenidos para cada valor de k_v mediante la Figura 10. En ambos casos para cada estado ocupado por un obstáculo se le asigna un color en función del número de visitas, correspondiéndole a aquellos menos visitados colores cercanos al blanco y a los más visitados cercanos al azul. Comparando lo obtenido, se puede observar como en el caso de k_v igual a 1 (Figura 10(a)) los estados más visitados se encuentran en la diagonal que corresponde a la trayectoria del agente, observándose en contadas ocasiones algunos que se encuentren fuera de ella. Sin embargo para el valor de 30000 (Figura 10(b)) se muestra como los estados visitados ya en el caso anterior son visitados un mayor número de veces, y en el resto se aprecia también una considerable mejora, destacando la que aparece en las zonas cercanas al destino.

Evaluando los modelos para los valores de 1 y 30000 se obtienen unos errores relativos del 86 % y 84 %, lo cual muestra que la exploración para las trayectorias y el entorno fijo sobre el que se experimenta no supone una mejora considerable sobre la ejecución del segundo agente. Este fenómeno se podría explicar teniendo en cuenta que en la trayectoria planteada para el segundo agente (2-B en Figura 5) las zonas más críticas serían aquellas en torno a los "sectores" donde se encuentran los estados inicial y final, y para ambos valores de k_v esas zonas se han explorado una cantidad de veces similar, tal y como se observa en la Figura 10. En el caso que aquí se planteaba se partía de la condición de que las salidas y destinos de los agentes estaban asignadas arbitrariamente, sin tener en cuenta la distribución de los obstáculos en el entorno, de manera que un mejor conocimiento del entorno a la hora de definir las trayectorias conduciría a obtener una exploración más orientada a la ejecución de éstas. Sin embargo, este hecho no resta importancia a la necesidad de explorar, también a efectos de conocer mejor el comportamiento del agente para poder orientar la exploración en otros problemas similares.



(a) Para $k_v = 1$



(b) Para $k_v = 30000$

Figura 10: Frecuencia de exploración de los estados con obstáculos

6. Implementación en una plataforma real

Como punto y final de este estudio se ha realizado la implementación del algoritmo *Dyna-Q* sobre una plataforma real compuesta por varios robots móviles dentro de un entorno, para evaluar los resultados obtenidos y abarcar las limitaciones que puedan existir. Se ha optado por realizarlo dentro de una plataforma ya existente y sobre la que actualmente se sigue trabajando, desarrollada dentro del Área de Ingeniería de Sistemas y Automática. Esta plataforma consta de unos robots móviles que funcionan con un procesador basado en la arquitectura *Arduino* y que están habilitados para recibir información mediante conexión Wi-Fi desde un ordenador central, el cual a su vez recibe una vista cenital del entorno en el que se mueve el agente mediante una cámara, y tras interpretarla orienta al agente desde su posición actual hasta su destino. En la Figura 11 se muestra una imagen de la plataforma durante la ejecución de una trayectoria.

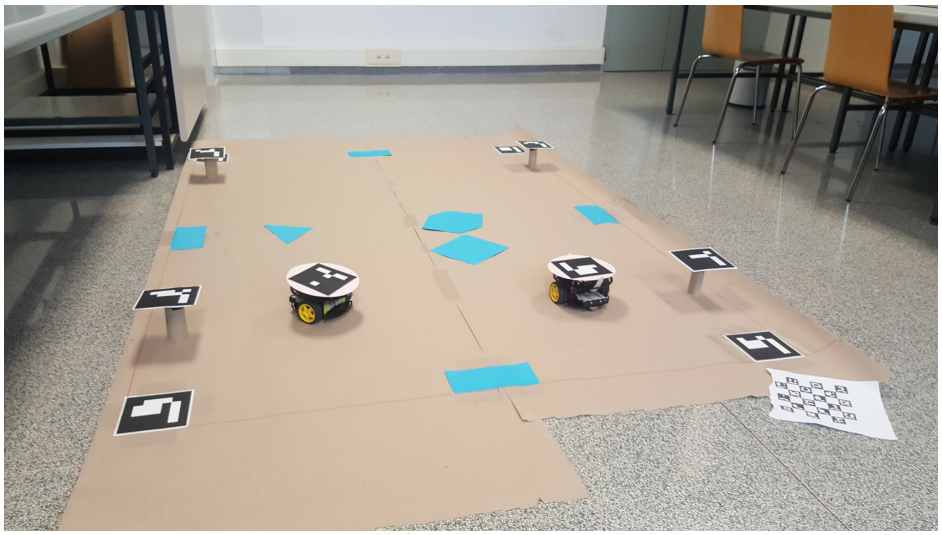


Figura 11: Plataforma multirobot durante una ejecución

El marco en el que se ha desarrollado la plataforma y se ha diseñado su estructura se basa en que el ordenador central sea el que realice la toma de decisiones y los agentes los que reciben la decisión y la ejecutan, un enfoque contrario al que se ha planteado a lo largo de éste estudio, donde los propios agentes deciden y ejecutan sus propias acciones de forma autónoma. Esto requiere que sea necesario adecuar algunos aspectos del problema a estas circunstancias dadas, de manera que se asemeje lo máximo posible, a nivel de

funcionamiento interno, a una estructura creada para este problema en particular. En primer lugar, al agente se le ha dotado inicialmente de un modelo interno con el conjunto de estados y su adyacencia, así como de las acciones que puede realizar, todo ello tal cual se había especificado en el planteamiento del problema. Su capacidad de detección, al carecer de un sistema de sensores integrado, se realizará mediante la información recogida por la cámara cenital en los estados circundantes al que se encuentre en cada momento el agente, y ésta se transmitirá al agente, que se encargará de su gestión. La comunicación que se plantea en la extensión al caso multiagente cuando uno de ellos alcanza un destino al resto se realizará a través del ordenador que detectará cuando un agente haya llegado a su destino, y se lo transmitirá al resto de agentes, al definirse la conexión Wi-Fi unidireccionalmente del ordenador a los robots, y no entre ellos.

A lo largo de este estudio se han comentado principalmente dos limitaciones físicas que podrían dificultar o incluso imposibilitar la implementación del *Dyna-Q* en una plataforma real, como son la memoria y el procesamiento, y se ha tratado su posible resolución mediante una ejecución eficiente del algoritmo en base a sus parámetros. La arquitectura basada en *Arduino* que emplean los agentes muestra, a priori, un nivel de procesamiento adecuado, radicando principalmente su inconveniente en la memoria dinámica, ocupada por aquellas variables necesarias para la ejecución. Quizás por ello, en este caso el ajuste de parámetros no se haga tan necesario para determinar la implementación ya que, tal y como se ha tratado en este trabajo, se ha orientado más como método para conseguir una ejecución más eficiente mediante un menor uso de recursos de procesamiento, al no considerarse el problema de la memoria como extremadamente crítico. De esta forma, se buscará en esta implementación una ejecución más óptima en cuanto al volumen de memoria interna usada, mediante la definición y uso de variables y la estructuración del código adecuados.

Por el momento se ha realizado un código que simula la toma de decisiones del agente sobre un entorno definido dentro de él, similar al existente en la plataforma que consta de 40 estados (5×8), y se ha evaluado su ejecución. El código probado y ejecutado incluye sólo la parte de Aprendizaje por Refuerzo del *Dyna-Q* y se ha incluido en el Apéndice F. Con él se ha obtenido un uso de la memoria dinámica por parte de las variables globales de 1182 bytes, que representa un 46 % de los 2560 disponibles, lo que se considera un buen resultado teniendo en cuenta que la ocupación de memoria será aproximadamente la misma cuando se incluya la parte de Planificación, ya que las variables que tienen más peso, como pueden ser $Q(s, a)$ o el vector en el que se contabilizan las visitas, ya están definidas y se usan en ambas

partes por igual. En cuanto al procesamiento, la toma de decisiones dentro de un estado consume tiempos por debajo de la décima de segundo, siendo la ejecución total del orden de unos pocos segundos, demostrándose así que el procesamiento no supone un inconveniente en la implementación.

7. Conclusiones y observaciones

Respecto a los resultados obtenidos en la fase de simulación cabe destacar que las interpretaciones y conclusiones a nivel general son aplicables sobre problemas de la misma naturaleza que el que se ha expuesto aquí, e incluso se podrían utilizar como base en los planteamientos de otros distintos. Pero a nivel particular de valores específicos de los parámetros, sólo se puede asegurar su utilidad en entornos que presenten el mismo número de estados y adyacencia entre ellos. Pese a esto, mediante todo el proceso de simulación se ha establecido una metodología y se han creado estructuras de gestión de experimentos y datos que podrían adaptarse para ser utilizadas en caso de evaluar otros entornos, algoritmos o problemas.

A nivel de aplicación de los resultados obtenidos en el caso de un agente sobre los casos multiagente, se ha demostrado que mediante P y γ se consigue un buen ajuste de las ejecuciones en todos los tamaños del mapa para 2 agentes. Para los casos con 3, aunque para grandes tamaños de mapa se ha obtenido una ejecución en torno al 25 % a nivel general los resultados no se adecúan a una buena ejecución. Esto invita a realizar un análisis más profundo sobre el caso de 3 agentes (o más) y establecer criterios más adecuados para caracterizarlos y poder usar así de forma más adecuada la utilización de los resultados del caso de un agente.

Lo referente a la parte de exploración ha permitido obtener ciertas conclusiones sobre el comportamiento de los agentes y la utilidad del conocimiento previo del mapa en función del planteamiento de su trayectoria y del entorno en el que se desarrolle. Dentro de esta sección también se podrían plantear la resolución de tipos de problema más ligados a la exploración y la evaluación de otros parámetros o políticas para ello.

La utilización de la media y el error relativo para la interpretación de los resultados ha resultado ser efectiva, pero se ha observado que presentan una serie de limitaciones. Con la media, todos los resultados se ponderan de la misma forma sobre el total, de manera que aquellos datos, bien erróneos o bien bajo unas ciertas situaciones poco frecuentes y representativas (*outliers*), pueden hacer que la media se desplace excesivamente de su valor esperado, más aún si se toman datos sobre un número limitado de casos. Esto podría explicar porque algunos resultados no se ceñían de forma estricta a los comportamientos esperados, aunque por lo general en las experimentaciones realizadas apenas se dan estas situaciones. Una posible medida sustitutiva sería la realización de una ponderación que asignase un menor peso a los valores excesivamente alejados del resto. En cuanto al error relativo, aunque la

medida no presenta inconvenientes (sin tener en cuenta que se basa en la media de movimientos) habla sobre una ejecución en términos de eficiencia de las trayectorias, al compararla con su tamaño mínimo. Esta perspectiva podría reconsiderarse y, por ejemplo, tomar como medida el error absoluto que representa el número de movimientos por encima del mínimo, si se desea establecer un criterio común independiente del tamaño de la trayectoria.

Sobre la implementación en una plataforma real, aunque no se haya podido realizar por completo, se ha conseguido plantear una aproximación que en un principio parece salvar la limitación de memoria que presentan los robots. Además, también se ha planteado una adaptación del problema de acuerdo al funcionamiento actual de la plataforma. Por estos motivos se seguirá trabajando en esta última parte.

Finalmente, y con respecto a los objetivos planteados al comienzo de este trabajo se ha conseguido lo siguiente:

- Se ha abordado el problema planteado mediante la correcta adaptación del algoritmo *Dyna-Q* a la naturaleza de éste, trabajando en la plataforma *Matlab*, e implementando lo obtenido dentro de la librería *Robotic Motion Toolbox*.
- El planteamiento de una versión extendida del *Dyna-Q* planteada en este trabajo ha demostrado resolver el problema de una forma eficiente, aunque será necesario un estudio más profundo de su ejecución.
- La experimentación ha arrojado conclusiones relevantes en el caso de un único agente sobre la influencia de los parámetros en su comportamiento, y los resultados obtenidos han resultado ser efectivos al aplicarlos sobre casos con 2 agentes.
- Se han podido abordar las limitaciones de la plataforma sobre la que se ha planteado la implementación de los métodos descritos en este estudio.

Referencias

[1] V. Mnih et al., *Human-level control through deep reinforcement*. Nature Science Journal, Feb. 2015.

[2] S. Levine et al., *Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection*. International Conference on Robotics and Automation, 2017.

[3] R.S. Sutton y A.G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, Second Edition, 2012.

[4] C.J.C.H Watkins, *Learning from Delayed Rewards*. PhD. Thesis, Cambridge University, 1989.

[5] G. Weiß, *An Architectural Framework for Integrated Multiagent Planning, Reacting, and Learning*. Intelligent Agents VII Agent Theories Architectures and Languages (ATAL), 2000.

[6] R. González y C. Mahulea y M. Kloetzer, *A Matlab-Based Interactive Simulator for Mobile Robotics*. IEEE Conference on Automation Science and Engineering, 2015.

[7] K.P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

Apéndice A Función *Dyna-Q* en *Matlab*

```
%% ALGORITMO DYNA_Q

% >Adaptado para cualquier numero de agentes.

% >Permite tanto la inicializacion del agente con un
% modelo existente como la obtencion del que resulta
% de su ejecucion


function [it,model] = Dyna_Q_model(model,dest_no,rob_no,...
                      env_bounds,obstacles,C,adj,s,d,...
                      mid_X,mid_Y,r_goal,rat_goal_trans,...
                      n_planning,m,visitf,prec,disc)




%------------------------------------------------------------
%% Parametros del algoritmo y entorno

r_object=-r_goal;
n_loops=rob_no*10000;  % Evitar ejecuciones infinitas
it=0;

r_trans=(-1)*(r_goal/rat_goal_trans);
visit_RL=1;
visit_pl=visit_RL/visitf;

alfa=1;

termpl=0;

prec_x=(env_bounds(2)-env_bounds(1))/prec;
prec_y=(env_bounds(4)-env_bounds(3))/prec;

% Maxima area ocupada por un estado para considerarlo no
% alcanzable
perc=0.05;
```

```
%-------------------------------------------------------------
%% Inicializacion de las variables

% Inicializacion de matriz Q, que funcionara como matriz
% Q(s,a), recogera las visitas realizadas en cada estado
% y la adyacencia entre ellos

n_st=size(adj,1);
Q_ind={};

a=1;
for i=1:n_st
    for j=1:n_st
        if (i~=j) && (adj(i,j)==1)
            Q_ind{1,i}(a,1)=j;
            Q_ind{1,i}(a,2)=0;
            Q_ind{1,i}(a,3)=0;
            a=a+1;
        end
    end
    a=1;
end
Q={};
visit=zeros(n_st,1);
for k=1:rob_no
    for dt=1:dest_no
        Q{k}(dt,:)=Q_ind;
    end
    for v=1:n_st
        Q{k}{dest_no+1,v}=0;
    end

end


% Matriz robmat registra los estados inicial, actual y
% siguiente
robmat={};
for k=1:rob_no
    robmat{k}(:,1)=[s(1,k);s(1,k)];
end


% Matrices que determinan los destinos alcanzados y por
% cual de los agentes ha sido alcanzado
term=zeros(1,rob_no);
reached=zeros(1,dest_no);
reachedby=zeros(1,dest_no);

```

```
%-------------------------------------------------------------
%% Algoritmo

% Mientras haya algun agente que no haya alcanzado
% un destino
while sum(term)<rob_no && it<=n_loops

    % k agentes
    for k=1:rob_no

        if ~term(k)

        current=robmat{k}(2,1);
        Q{k}{dest_no+1,current}=Q{k}{dest_no+1,current}...
                                +visit_RL;


        % Eleccion de accion segun el estado
        % actual -> e-greedy
         chi=rand(1,1);
         epsi= m/Q{k}{dest_no+1,current};

         if chi<=epsi
                % Accion de forma aleatoria-> Explorar
                act=randsample(Q{k}{1,current}(:,1),1);
           else
                % Mejor accion-> Maximizar recompensa
                Qmap=zeros(size(Q{k}{1,current},1),1);

                % Evaluar sobre el total de destinos no
                % alcanzados -> Extension al caso multiagente
                for dt=1:dest_no
                    if ~reached(dt)
                        Qmap=Qmap+Q{k}{dt,current}(:,2);
                    end
                end
                [mxm,idx]=max(Qmap);
                rep=length(find((Qmap<=mxm+eps*1000) & ...
                                (Qmap>=mxm-eps*1000)));
                if rep==4
                    % Cuando no existe una accion maxima
                    % (cualquier accion conlleva la misma
                    % recompensa) se elige de forma
                    % aleatoria
                    act=randsample(Q{k}{1,current}(:,1),1);
                else
                    act=Q{k}{1,current}(idx,1);
                end
           end
```

```
144
145             % Deteccion de objetos en zonas
146             % circundantes -> Actualizacion del modelo
147
148             [model] = sensing(k,model,current,mid_X,mid_Y,...
149                               C,prec_x,prec_y,obstacles);
150
151
152             % Proximo estado
153             [nxt] = nxt_state(C,prec_x,prec_y,perc,...
154                               obstacles,current,act);
155             robmat{k}(3,1)=nxt;
156
157             % Evaluacion de las recompensas -> Para
158             % todos los destinos no alcanzados
159             R=zeros(dest_no,1);
160             for dt=1:dest_no
161                 if ~reached(dt)
162                     if nxt==d(dt)
163                         % Alcanzar el destino
164                         R(dt)=r_goal;
165                         term(k)=1;
166                     elseif nxt==current
167                         % Movimiento hacia un estado ocupado
168                         R(dt)=r_object;
169                     else
170                         % Transicion a otro estado
171                         R(dt)=r_trans;
172                     end
173                 end
174             end
175
176
177             % Actualizacion Q-learning
178             for dt=1:dest_no
179
180                if ~reached(dt)
181                last=size(Q{k}{dt,current},1);
182                for j=1:last
183                if Q{k}{dt,current}(j,1)==act
184
185                Q{k}{dt,current}(j,3)=
186                         Q{k}{dt,current}(j,3)+visit_RL;
187
188                [mxm,idx]=max(Q{k}{dt,nxt}(:,2));
189
190                Q{k}{dt,current}(j,2)=Q{k}{dt,current}(j,2)+...
191                     alfa*(R(dt)+disc*(Q{k}{dt,nxt}(idx,2))-...
192                     Q{k}{dt,current}(j,2));
```

```
193                 end
194                 end
195                 end
196
197             end
198
199             % Evaluacion de si el agente
200             for dt=1:dest_no
201                 if ~reached(dt)
202                     if nxt==d(dt)
203                         reached(dt)=1;
204                         reachedby(dt)=k;
205                     end
206                 end
207             end
208
209
210             % Planificacion
211
212             if ~term(k)
213
214                 robmatpl=robmat;
215
216                 for p=1:n_planning
217
218                 currentpl=robmatpl{k}(2,1);
219                 Q{k}{dest_no+1,currentpl}=
220                       Q{k}{dest_no+1,currentpl}+visit_pl;
221
222                 % Toma de decisiones aleatorias
223                 actpl=randsample(Q{k}{1,currentpl}(:,1),1);
224                 nxtpl=actpl;
225
226                 % Evaluacion de las recompensas
227                 R=zeros(dest_no,1);
228                 for dt=1:dest_no
229                     if ~reached(dt)
230                         if nxtpl==d(dt)
231                             % Alcanzar el destino
232                             R(dt)=r_goal;
233                             termpl=1;
234                         else
235                             findd= model{k}==nxtpl;
236                             if sum(findd)==1
237                                 %Movimiento hacia un
238                                 %estado ocupado
239                                 R(dt)=r_object;
240                                 nxtpl=currentpl;
241                             else
```

```
242                                     %Transicion a otro estado
243                                     R(dt)=r_trans;
244                                 end
245                             end
246                         end
247                     end
248
249                     % Actualizacion de Q-learning
250                     for dt=1:dest_no
251                         if ~reached(dt)
252                         last=size(Q{k}{dt,currentpl},1);
253                         for j=1:last
254                         if Q{k}{dt,currentpl}(j,1)==actpl
255
256                         Q{k}{dt,currentpl}(j,3)=
257                             Q{k}{dt,currentpl}(j,3)+visit_pl;
258
259                         [mxm,idx]=max(Q{k}{dt,nxtpl}(:,2));
260
261                         Q{k}{dt,currentpl}(j,2)=...
262                           Q{k}{dt,currentpl}(j,2)+alfa*...
263                           (R(dt)+disc*Q{k}{dt,nxtpl}(idx,2)-...
264                           Q{k}{dt,currentpl}(j,2));
265                         end
266                         end
267                         end
268                     end
269
270                     robmatpl{k}(2,1)=nxtpl;
271
272                     end %p=1:n_planning
273
274                 end %if ~term(k)planning
275
276                 robmat{k}(2,1)=nxt;
277                 termpl=0;
278
279
280             end %If ~term(k)
281
282         end   %For k=1:rob_no
283         it=it+1;
284
285     end %While sum(term)...
286
287
288     end
```

Apéndice B Función *Sensing* en *Matlab*

```
%% ALGORITMO SENSING

% >Simula la capacidad de deteccion del agente en los
% estados circundantes al que se encuentra

% >Actualiza el modelo del agente


function [model] = sensing(k,model,currentst,mid_X,mid_Y,...
                           C,prec_x,prec_y,obstacles)

ev_states=zeros(8,1);
n_st=size(mid_X,1);

% Se resuelve de forma geometrica, diviendose el espacio
% en los 8 estados circundantes al estado
% actual 'S' del agente
%                   -------------
%                   | 5 | 4 | 7 |
%                   -------------
%                   | 1 | S | 2 |
%                   -------------
%                   | 6 | 3 | 8 |
%                   -------------

%-------------------------------------------------------------
% Estados 1 y 2
ev_x=[];
for j=1:n_st
    val=mid_X(currentst,j);
    if val~=0
        ev_x=[ev_x; j val];
    end
end

[minn,imin]= min(ev_x(:,2));
[maxx,imax]= max(ev_x(:,2));

if abs(minn-C{currentst}(1,1))<eps*1000
    ev_states(1)=ev_x(imin,1);
end

if abs(maxx-C{currentst}(1,2))<eps*1000
    ev_states(2)=ev_x(imax,1);
end
```

```
%-------------------------------------------------------------
% Estados 3 y 4
ev_y=[];
for j=1:n_st
    val=mid_Y(currentst,j);
    if val~=0
        ev_y=[ev_y; j val];
    end
end

[minn,imin]= min(ev_y(:,2));
[maxx,imax]= max(ev_y(:,2));

if abs(minn-C{currentst}(2,2))<eps*1000
    ev_states(3)=ev_y(imin,1);
end

if abs(maxx-C{currentst}(2,3))<eps*1000
    ev_states(4)=ev_y(imax,1);
end

%-------------------------------------------------------------
% Estados 5,6,7 y 8

ar=5;
ab=6;
for r=1:2
    ev=[];
    if ev_states(r)~=0
        for j=1:n_st
            val=mid_Y(ev_states(r),j);
            if val~=0
                ev=[ev; j val];
            end
        end

        [minn,imin]= min(ev(:,2));
        [maxx,imax]= max(ev(:,2));

        if abs(minn-C{ev_states(r)}(2,2))<eps*1000
            ev_states(ab)=ev(imin,1);
        end
        ab=ab+2;

        if abs(maxx-C{ev_states(r)}(2,3))<eps*1000
            ev_states(ar)=ev(imax,1);
        end
        ar=ar+2;
    end
```

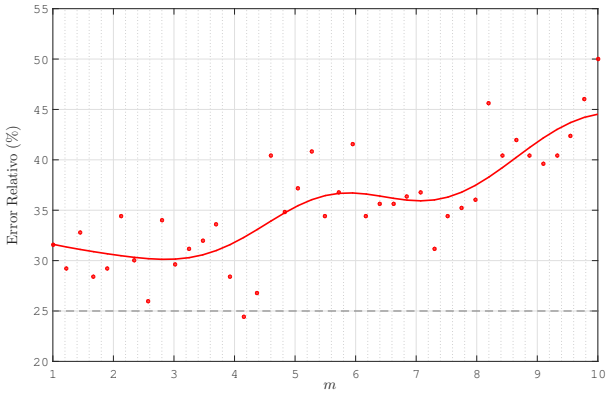
```
95  end
96
97
98  %-----------------------------------------------------------
99  % Evaluar la ocupacion de los estados
100
101 for i=1:size(ev_states,1)
102     findd= model{k}==ev_states(i);
103     if (ev_states(i)~=0) && (sum(findd)==0)
104         [bool] = obj_state(C,prec_x,prec_y,...
105                            obstacles,ev_states(i));
106         if bool
107             model{k}=[model{k};ev_states(i)];
108         end
109     end
110 end
111
112
113 end
```

Apéndice C Función *Rectangular Decomposition* en *Matlab*

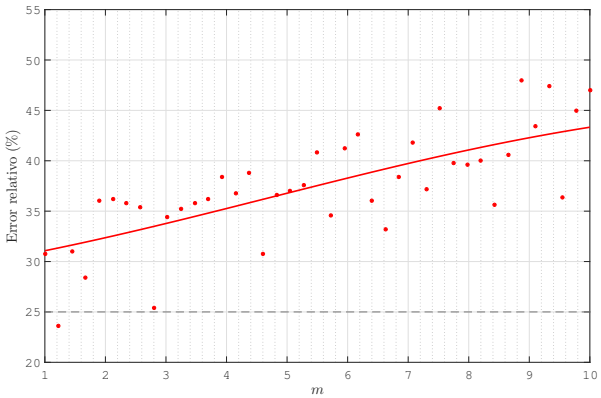
```
%% DESCOMPISICION RECTANGULAR UNIFORME DEL ENTORNO

%>En la descomposicion no se tiene en cuenta los objetos,
% y tambien se obtienen las adyacencias entre los estados.

%>Se obtienen estados rectangulares con las mismas
%dimensiones


function ...
    [C,adj]=rectangular_decomposition(env_bounds,varargin)
if nargin==1
    prec=8;
else
    prec=varargin{1};
end

% Precision en los ejes del mapa
prec_x=(env_bounds(2)-env_bounds(1))/prec;
prec_y=(env_bounds(4)-env_bounds(3))/prec;


C={};
C = divide_map(C,prec_x, prec_y, env_bounds(1), ...
    env_bounds(2), ...
                env_bounds(3), env_bounds(4));
% Adyacencia entre
k=length(C);
adj=sparse(eye(k));

% Permite obtener


% La obtencion de la adyacencia se basa en la
%representacion-H de los rectangulos
Ak=[0 -1;-1 0;0 1;1 0];
for i=1:(k-1)
    bi=[C{i}(2,1) C{i}(1,1) -C{i}(2,3) -C{i}(1,2)]';
    for j=(i+1):k
        bj=[C{j}(2,1) C{j}(1,1) -C{j}(2,3) -C{j}(1,2)]';
        V=cddmex('extreme',struct('A',[Ak;Ak],'B',[-bi;-bj]));
        if size(V.V,1)==2
            adj(i,j)=1;
```

```
42              adj(j,i)=1;
43          end
44      end
45  end
46
47  % Organizar los rectangulos para que formen
48  % una figura convexa
49  for i=1:length(C)
50      ch_in=convhull(C{i}(1,:),C{i}(2,:));
51      C{i}=C{i}(:,ch_in(1:length(ch_in)-1));
52  end
53
54  end
55
56  %Sub programa de subdivision de los rectangulos
57  function C = divide_map(C,prec_x, prec_y, x1,x2,y1,y2)
58
59  for y=y1:prec_y:(y2-prec_y)
60      ya=y;
61      yb=y+prec_y;
62      for x=x1:prec_x:(x2-prec_x)
63          xa=x;
64          xb=x+prec_x;
65          C{end+1}=[xa xb xb xa; ya ya yb yb];
66      end
67  end
68
69  end
```

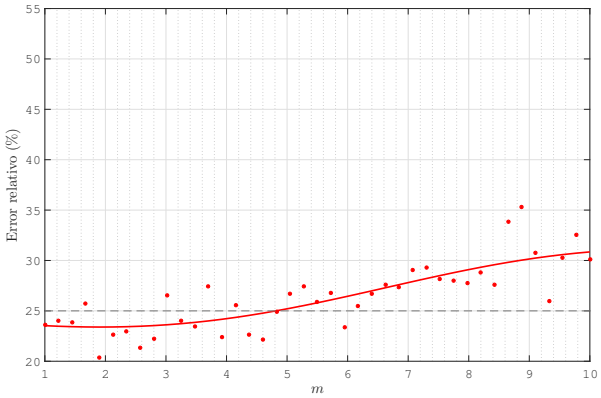
Apéndice D Resultados del parámetro m



(a) Tamaño 5



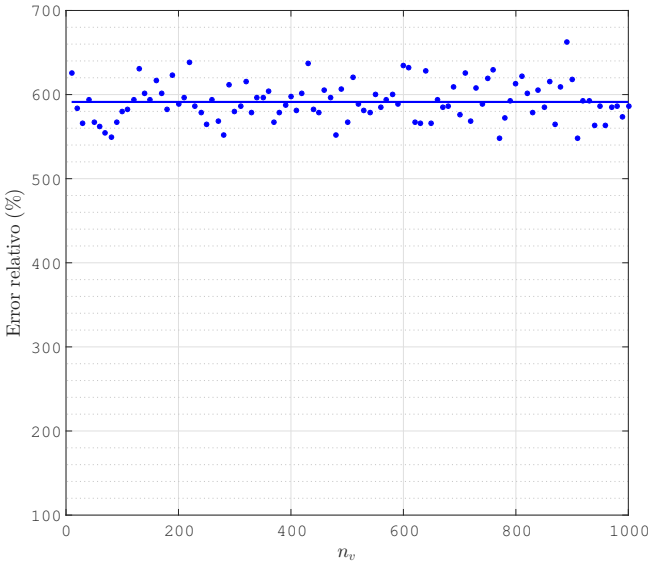
(b) Tamaño 10



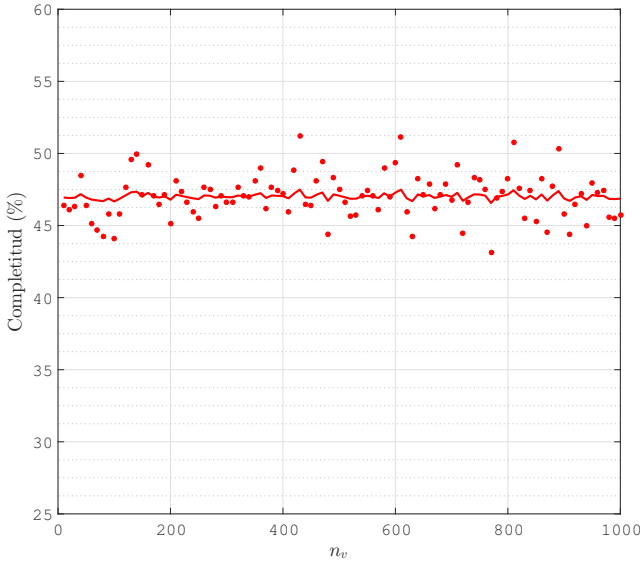
(c) Tamaño 15

Figura 12: Error relativo (%) frente a m para distintos tamaños de trayectoria

Apéndice E Resultados del parámetro n_v



(a) Error relativo (%) frente a n_v



(b) Completitud del modelo(%) frente a n_v

Figura 13: Completitud del modelo y error relativo en función de n_v

Apéndice F Código implementado en Arduino

```
1
2  /* ALGORITMO DYNA-Q -> Solo proceso de Aprendizaje por ...
       Refuerzo
3
4    -El mapa se encuentra dentro del algoritmo ->
5     -> Variable objeto con los estados ocupados
6
7
8  */
9
10 ////////////////////////////////////////////////////////////
11 // SETUP
12
13 void setup() {
14   //Serial1.begin(9600); //Conexion Wifi
15 }
16
17 ////////////////////////////////////////////////////////////
18 // VARIABLES GLOBALES
19
20 // Entorno y robot
21
22 int n_dt=1;
23
24
25 // Recoge las acciones posibles dentro de casa estado (0-39)
26 boolean actmat[4][40]=
27 {{0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ...
       ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ...
       ,1 ,1 ,1},//^
28
29 {1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ...
       ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,0 ,0 ,0 ,0 ...
       ,0 ,0 ,0},//v
30
31 {0 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ...
       ,1 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,1 ...
       ,1 ,1 ,1},//<
32
33 {1 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ...
       ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,1 ,1 ...
       ,1 ,1 ,0}};//>
34
35
36 boolean reached[1]={false}; // Tamano = Numero de destinos
```

```
37 boolean term=false; // Indica final de trayectoria del robot
38
39 // Estados inicial y final
40 int start[2]={1,1};
41 int dest[1][2]={2,7};// Tamano = Numero de destinos
42
43 // Modelo interno del agente
44 boolean model[40]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
45                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
46 //          0-> Sin obstaculo    1->Con obstaculo
47
48 int modelidx=-1;
49 int obj[7]={26,11,19,27,5,13,21};
50
51
52 // Parametros y variables algoritmo
53
54 long r_goal=1000000;
55 long r_trans=-1000;
56 long r_object=-r_goal;
57 int alfa=1;
58 int n_planning = 5000;
59 float visit_RL = 1;
60 float visit_pl = 1;
61 float disc = 5000;
62 float m = 1;
63
64 //Vector visitas
65 long visit[40];
66
67 //Matriz Q
68 float Q[4][40];
69 int current[2];
70 int nxt[2];
71
72 int it=0;
73
74
75 ////////////////////////////////////////////////////////////
76
77 // Subprograma para el calculo de chi
78 double randomDouble(double minf, double maxf)
79 {
80   return minf + random(1UL << 31) * (maxf - minf) / (1UL ...
        << 31);  // use 1ULL<<63 for max double values)
81 }
82
83
84
```

```
85 ////////////////////////////////////////////////////////////
86 // ALGORITMO
87
88
89 void DynaQ(){
90   //delay(10);
91   if (!term) {
92
93     if (it==0){
94       current[0]=start[0];
95       current[1]=start[1];
96     }
97     int cst=current[0]*8+current[1];
98
99     visit[cst]=visit[cst]+visit_RL;
100
101    // Politica e-greedy
102    float chi= randomDouble(0.01, 1.00);
103    float epsi =m/visit[cst];
104    int possible=false;
105    int i,f,act,dt;
106    float maxi;
107    if (chi<=epsi){
108      while (!possible){
109        act =random(4);
110        if (actmat[act][cst]==1){
111          possible=true;
112        }
113      }
114
115    }
116    else{
117      float Qmap[4];
118      for (dt=0;dt<=(n_dt-1);dt++){
119        for (i=0;i<=3;i++){
120          if (reached[dt]==false){
121            f=4*dt+i;
122            Qmap[i]=Qmap[i]+Q[f][cst];
123          }
124        }
125      }
126      maxi=0;
127      for (i==0;i<=3;i++){
128        if ((Qmap[i]>=maxi)&&(actmat[i][cst]==1)){
129          maxi=Qmap[i];
130          act=i;
131        }
132      }
133      if (actmat[act][cst]==0){
```

```
134           while (!possible){
135             act =random(4);
136             if (actmat[act][cst]==1){
137               possible=true;
138             }
139           }
140         }
141 
142       }//if (chi...
143 
144 
145 
146       // Actualizacion del modelo
147 
148       int x,y,j;
149       int idx=0;
150       int ev;
151 
152       for (j=-1;j<=1;j++){ //x
153         for (i=-1;i<=1;i++){//y
154           x=current[0]+j;
155           y=current[1]+i;
156           if ((x>=0) && (y>=0)){
157             ev=x*8+y;
158             for (idx=0;idx<=6;idx++){
159               if ((ev==obj[idx])&&(model[ev]==0)){
160                 model[ev]=1;
161               }
162             }
163           }// for idx
164         }// for i
165       }// for j
166 
167 
168       // Estado siguiente en funcion de la decision tomada y ...
             de si el estado //permite dicha accion
169 
170       if (act<=1){
171         nxt[1]=current[1];
172         if (act==0){
173           nxt[0]=current[0]-1;
174           //Arriba
175         }
176         else{
177           nxt[0]=current[0]+1;
178           //Abajo
179         }
180       }
181       else{
```

```
182         nxt[0]=current[0];
183         if (act==2){
184           nxt[1]=current[1]-1;
185           //Izquierda
186         }
187         else{
188           nxt[1]=current[1]+1;
189           //Derecha
190         }
191       }
192
193
194
195
196       //Evaluacion de si la proxima accion conduce a
197       //un obstaculo
198
199       int nst=nxt[0]*8+nxt[1];
200       if (model[nst]==1){
201         nxt[0]=current[0];
202         nxt[1]=current[1];
203         nst=nxt[0]*8+nxt[1];
204         ;
205       }
206
207
208
209       //Evaluacion de las recompensas
210       float R[n_dt];
211       for (dt=0;dt<=(n_dt-1);dt++){
212         if (reached[dt]==false){
213           if ((current[0]==nxt[0])&&(current[1]==nxt[1])){
214             // Se accede a un estado ocupado por
215             //un obstaculo
216             R[dt]=r_object;
217           }
218           else if((dest[dt][0]==nxt[0])&&...
219                   (dest[dt][1]==nxt[1])){
220             R[dt]=r_goal;
221             term=true;
222             reached[dt]=true;
223           }
224           else{
225             R[dt]=r_trans;
226           }
227         }
228       }
229       int maxidx;
230
```

```
231
232
233
234       //Actualizacion Q-learning
235       for (dt=0;dt<=(n_dt-1);dt++){
236         if (reached[dt]==false){
237           f=4*dt;
238           maxi=0;
239           for (i==0;i<=3;i++){
240             if (Q[i][nst]>=maxi){
241               maxi=Q[i][nst];
242               maxidx=i;
243             }
244           }
245           Q[f+act][cst]=Q[f+act][cst]+...
246           alfa*(R[dt]+disc*Q[f+i][nst]-Q[f+act][cst]);
247         }
248       }
249
250       //Actualizacion de estados
251       current[0]=nxt[0];
252       current[1]=nxt[1];
253       it++;
254
255     }// if !term
256   }
257
258
259   ////////////////////////////////////////////////////////////
260   // LOOP
261
262   void loop() {
263     DynaQ();
264   }
```