Universidad de Alcalá
Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Trabajo Fin de Máster

SOLUCIONES DE NAVEGACIÓN INTELIGENTE PARA PLATAFORMAS ROBÓTICAS ENTRENADAS EN ENTORNOS VIRTUALES

Autor: Eduardo Gutiérrez Maestro
Tutor: Roberto J. López Sastre

2019

Universidad de Alcalá

Escuela Politécnica Superior

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Soluciones de navegación inteligente para plataformas robóticas entrenadas en entornos virtuales

Autor: Eduardo Gutiérrez Maestro

Director: Roberto J. López Sastre

TRIBUNAL:

Presidente: D. Francisco Javier Acevedo Rodríguez

Vocal 1º: D. David Fernández Barrero

Vocal 2º: D. Roberto J. López Sastre

FECHA: 12/09/2019

Soluciones de navegación inteligente para plataformas robóticas entrenadas en entornos virtuales

Eduardo Gutiérrez Maestro

12 de septiembre de 2019

Wer kämpft, kann verlieren. Wer nicht kämpft, hat schon verloren.
"Quien lucha, puede perder. Quien no lucha, ya ha perdido."
Bertolt Brecht

Agradecimientos

Quería agradecer el apoyo de todas las personas que me han acompañado a lo largo de mi recorrido por la universidad, en especial a mis padres. Por su apoyo constante, su entrega y continua confianza en mi esfuerzo para sacar todos mis objetivos a delante. Agradecer también los consejos de mi hermano, sabiendo en todo momento que decir y cómo apoyar con tan sólo un: *"¡Edu, tú puedes!"*

Finalmente, quería agradecer el apoyo y la ayuda de mi director, Roberto, que durante toda esta etapa final me ha ayudado y guiado en aspectos tanto universitarios como profesionales. Agradecerle también la oportunidad brindada de adentrarme en el camino profesional que sigo y que espero seguir fuera de la universidad.

Índice general

Lista de figuras

Lista de tablas

Resumen

La navegación visual es la capacidad que tiene un agente autónomo de encontrar su camino en un entorno amplio y complejo basado únicamente en información visual. De hecho, es un problema fundamental en la visión por computador y la robótica. En este proyecto se propone un modelo basado en deep reinforcement learning que es capaz de navegar en una escena para alcanzar un objetivo *visual*, pero anticipando las posibles colisiones dentro del entorno. Técnicamente, se propone un modelo de tipo *map-less*, que sigue un método de reinforcement learning conocido como *actor-critic*, en donde la función de recompensa ha sido diseñada para evitar colisiones. Se expone una evaluación exhaustiva del modelo para el entorno virtual AI2-THOR, donde los resultados muestran que el modelo propuesto: 1) mejora el estado del arte en términos de número de pasos y de colisiones; 2) es capaz de converger más rápido que un modelo que no tiene en cuenta las colisiones, buscando únicamente el camino más corto; y 3) ofrece una interesante capacidad de generalización para alcanzar objetivos visuales que no han sido nunca vistos durante el entrenamiento.

Palabras clave: navegación visual, deep reinforcement learning, robótica, visión por computador

Abstract

Visual navigation is the ability of an autonomous agent to find its way in a large and complex environment based on visual information. It is indeed a fundamental problem in computer vision and robotics. In this project, it is proposed a deep reinforcement learning approach which is able to learn to navigate a scene to reach a given *visual* target, but anticipating the possible collisions with the environment. Technically, it is proposed a map-less-based model, which follows an actor-critic reinforcement learning method where the reward function has been designed to be collision aware. It is offered a thorough experimental evaluation of our solution in the AI2-THOR virtual environment, where the results show that the proposed method: 1) improves the state of the art in terms of number of steps and collisions; 2) is able to converge faster than a model which does not care about the collisions, simply searching for the shortest paths; and 3) offers an interesting generalization capability to reach visual targets that have never been seen during training.

Keywords: visual navigation, deep reinforcement learning, robotics, computer vision

Resumen Extendido

El desarrollo de aplicaciones autónomas es una tendencia cada vez más puntera en la industria actual. La tecnología avanza cada día más rápido, siendo un factor determinante para la creación de aplicaciones con autonomía que resuelvan tareas de la vida cotidiana, satisfaciendo las exigencias y comodidades del día a día. Un claro ejemplo es el sector de la robótica, que se ha aprovechado de este avance para desarrollar plataformas robóticas con cierto nivel de independencia para la resolución de tareas. El campo de la automoción es uno de los afectados por el desarrollo de este tipo de aplicaciones, con la ya presente llegada del coche autónomo, siendo capaz de tomar decisiones por el conductor. Otro de los campos es el sector industrial con la mejora de la automatización de los brazos robóticos, de manera que la interacción humano-máquina sea más segura.

En concreto, en este TFM se persigue la implementación de un modelo que resuelva el problema de la navegación visual para plataformas robóticas en entornos de interior. Este problema se define como la capacidad de una plataforma robótica de alcanzar un objetivo usando únicamente como información imágenes visuales capturadas por la plataforma. Por tanto, el objetivo principal de este proyecto consiste en aprender a mapear imágenes bidimensionales en acciones que serán ejecutadas en un espacio 3D. Para ello, se hace uso de una técnica de visión por computador conocida como *deep* RL. Éste método se hizo famoso con trabajos como [3], donde una máquina aprende a jugar a los juegos de ATARI mejor que un ser humano. Por otra parte, para facilitar el proceso de entrenamiento los datos son extraídos de un entorno virtual conocido como AI2-THOR [1], ya que la cantidad de datos que requieren este tipo de algoritmos es muy grande como para hacer capturas en escenarios reales, lo cual requiere un gran gasto de recursos y tiempo.

La filosofía perseguida durante este TFM busca el desarrollo de plataformas que se muevan como hacen los humanos: teniendo en cuenta el entorno y los objetos que les rodean. Más aún cuando se quiere dotar de significado a la tarea de navegación en sí, como puede ser ir en busca de un objeto para a continuación interactuar con él. El problema de navegación de plataformas robóticas ha sido tratado en gran profundidad durante los últimos años. La mayoría de soluciones propuestas son consideradas como métodos *map-based*, es decir, el robot necesita un mapa de la escena por dónde navega para que pueda tomar decisiones, como en [4, 5]. Otros métodos, en cambio, realizan una reconstrucción

del mapa del entorno sobre la marcha para navegar, por ejemplo [6, 7, 8]. Además de los mencionados, se pueden destacar métodos que hacen uso de directrices basadas en el conocimiento humano para la generación del mapa como ocurre en [9, 10]. Finalmente, se destacan los métodos conocidos como *map-less*. Éstos no necesitan ningún mapa, sin necesidad de ningún punto de referencia para alcanzar los objetivos marcados. El trabajo presentado en [2] sigue este método, el cual es tomado como modelo base de este TFM.

El modelo que se expone en el presente documento, sigue esta metodología *map-less*, presentando una solución para la navegación visual que combina los avances más recientes en *deep* y *reinforcement learning*. Este proyecto pretende impulsar la navegación robótica adoptando comportamientos humanos. Con este propósito, surge la siguiente pregunta: ¿cómo aprende un humano a moverse dentro de un entorno cerrado?

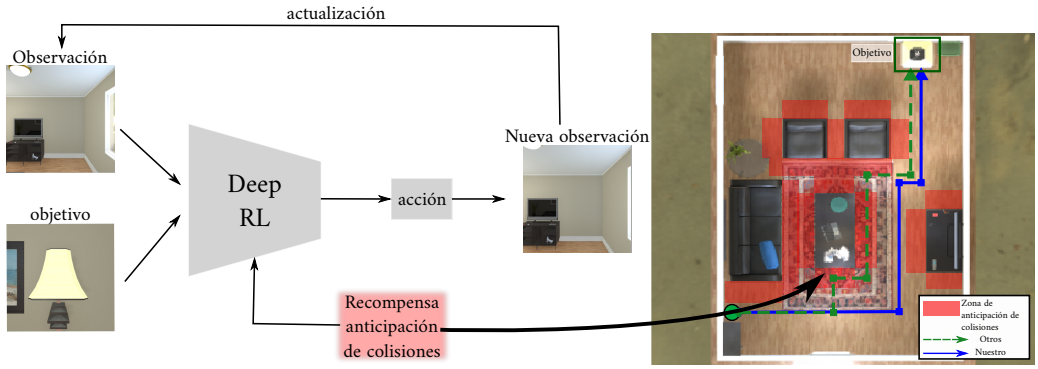


Figura 1: Idea para la navegación propuesta en el presente TFM.

La idea planteada en este proyecto queda reflejada en la figura 1. ¿Qué camino escogería un humano para alcanzar la lámpara? En este trabajo, se defiende la tendencia de los humanos a moverse evitando los obstáculos que puedan presentarse, anticipándolos. En otras palabras, los humanos se desplazan manteniendo un margen de separación con los posibles objetos del entorno que les rodean. Por tanto, el objetivo fijado en este TFM consiste en transmitir esta filosofía de anticipación de colisiones, de manera que un agente sea capaz de alcanzar objetivos realizando trayectorias con el menor número de pasos y colisiones posibles.

El desarrollo de este TFM se puede dividir en dos grandes bloques. El primer bloque está relacionado con el entorno virtual y la base de datos empleados en este proyecto. El segundo bloque consiste en la elaboración de un modelo basado en *deep* RL para lograr el objetivo de navegación marcado en el TFM.

Como ya se comentó previamente, la cantidad de datos necesarios para el entrenamiento de algoritmos basados en *deep* RL es inmensa, por ello se hace uso de entornos virtuales. En el caso de este proyecto, se hace uso de uno de los entornos más novedosos: AI2-THOR, caracterizado por sus escenas de alta similitud a imágenes reales y la posibilidad de interacción con los objetos que la forman. La solución de navegación propuesta

en este TFM se considera como discreta, es decir, el agente realiza movimientos discretos durante el proceso de navegación. Para simular esta navegación discreta, se introduce el concepto *mundo de cuadrícula*, donde la escena queda discretizada en diferentes celdas. En la figura 2 se muestra un ejemplo del proceso de discretización llevado a cabo para simular el tipo de navegación planteada en este proyecto.

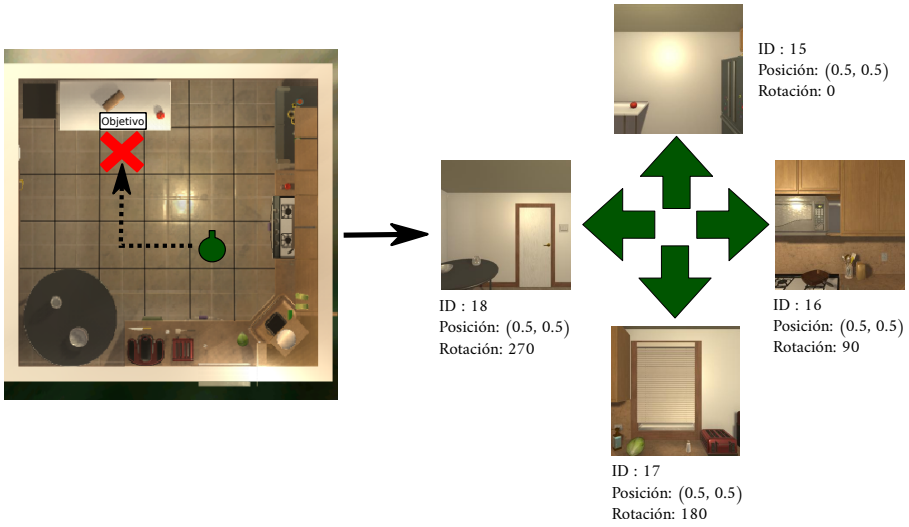


Figura 2: Ejemplo del proceso de discretización para simular el tipo de navegación planteada en el TFM.

Además, en la figura 2 se refleja de manera clara el objetivo del robot: navegar hacia la celda en la que se tomó la imagen del objetivo al que se desea navegar. Como se observa en la figura 2, el robot es capaz de ejecutar cuatro diferentes movimientos: moverse hacia delante, girar a derecha, girar a izquierda o desplazarse hacia detrás.

Una vez definido el problema de navegación discreta, se procede a introducir el modelo desarrollado para el presente TFM. Antes de entrar en detalle del modelo, es necesario definir los tres ingredientes básicos que componen cualquier algoritmo basado en RL, que son:

- **Espacio de acciones.** Define los diferentes movimientos que el agente es capaz de ejecutar en el entorno virtual.
- **Observaciones.** El agente está provisto con una cámara frontal. Cada observación representa un estado del entorno que el agente recibe cada vez que ejecuta una acción.
- **Función de recompensa.** En RL la función de recompensa es la manera de dar respuesta, con la que se mide el éxito o fracaso de la acción tomada por el agente. En esta función se introduce la filosofía para la anticipación de colisiones mencionada anteriormente. Para ello se dividen todos los posibles estados de una escena en cuatro grupos, asignando un valor de recompensa que puede ser positivo si el robot llega al

objetivo, o negativo si hay riesgo de colisión durante la navegación. De esta manera el agente aprende a navegar por aquellos estados que le permitan maximizar esta señal de recompensa.

La figura 3 muestra la arquitectura de red empleada en este proyecto. La primera parte de la red, las capas siamesas, es alimentada con vectores de características obtenidos del modelo de red ResNet-50 pre entrenado en la base de datos ImageNet [11]. Los parámetros de la ResNet-50 son *congelados* durante la fase de entrenamiento. Cada uno de los agentes (que entrena un objetivo) actualiza los pesos compartidos en la red global. En el lado derecho de la figura, se contemplan las redes específicas para cada escena, que dan como salida los valores de la política aprendida según la entrada visual para cada escena en particular. El método empleado para su entrenamiento es conocido como Asynchronous Advantage Actor-Critic (A3C) [12], el cual actualmente se encuentra entre los algoritmos más eficientes para resolver problemas basados en RL.

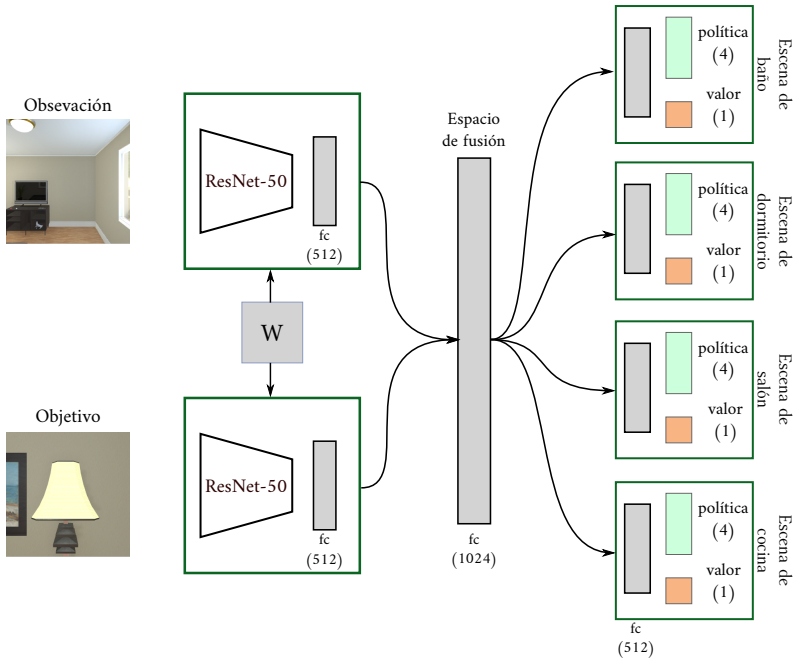


Figura 3: Modelo de arquitectura de red presentado en [2] y empleado en el TFM.

El algoritmo A3C destaca por su capacidad de ejecutar en paralelo distintas copias de la red deseada. En el caso de este proyecto, el algoritmo lanza una copia por cada objetivo al que se desea navegar. Esta copia se denominará *trabajador*. La ventaja que presenta este algoritmo reside en su capacidad de actualizar la información aprendida de los diferentes *trabajadores* de manera asíncrona. De este modo, se consiguen aprender las políticas de movimiento de manera mucho más rápida.

Para evaluar el modelo presentado en este proyecto, se emplearon dos métricas: 1) número de pasos y colisiones sufridas durante el episodio; 2) *puntuación de éxito*, que

consiste en el porcentaje de veces en los que el agente es capaz de llegar a su objetivo en un número de pasos inferior a 500, la cual únicamente fue empleada en los experimento de generalización que se expondrán posteriormente.

A continuación, se introducen al lector los principales experimentos realizados para validar el modelo expuesto en este TFM, pero se recuerda al lector que la evaluación llevada a cabo en este proyecto es realizada en comparativa con el modelo presentado por Zhu *et al.* [2], el cual es tomado como modelo base para este trabajo. El primer experimento consistió en evaluar la validez del modelo planteado en este proyecto. El objetivo del mismo es corroborar que el modelo que se plantea es capaz de reducir, es decir anticipar, el número de colisiones durante la navegación.

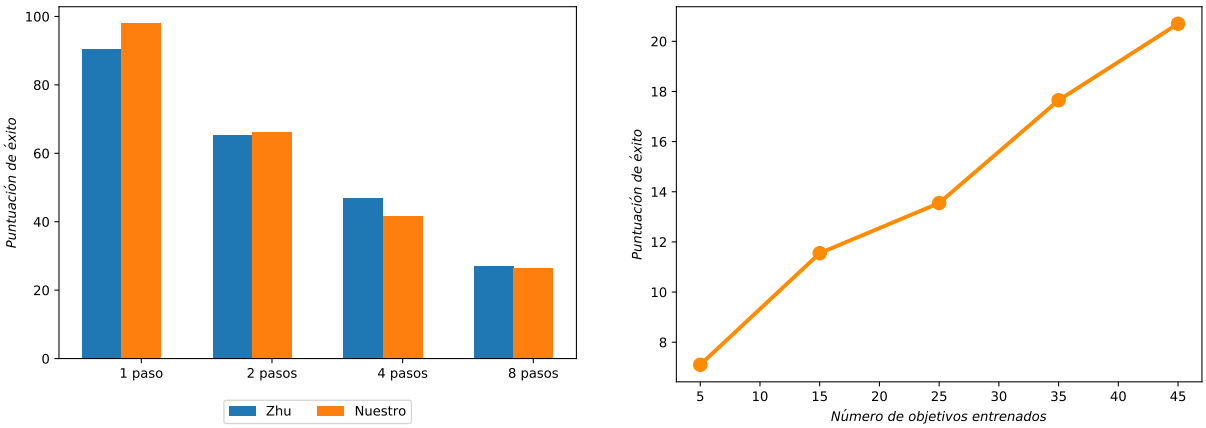
	Escena	Baño	Dormitorio	Salón	Cocina	Promedio
Zhu [2]	Pasos	7.17	14.82	15.2	21.38	14.7
	Colisiones	0.04	0.12	0.25	0.2	0.15
Nuestro	Pasos	7.33	14.81	14.9	20.83	**14.47**
	Colisiones	0.03	0.04	0.15	0.12	**0.082**

Tabla 1: Resultados para el experimento de navegación.

La tabla 1 muestra los resultados obtenidos. Contemplando el número de colisiones sufridas durante la navegación, uno puede apreciar que el modelo presentado en este TFM las reduce de manera considerable respecto al modelo base. Cabe resaltar que el modelo presentado en [2] no tiene en cuenta las colisiones durante el proceso de aprendizaje, optimizando el modelo para que el agente encuentre únicamente el camino más corto hasta su objetivo. Los resultados expuestos verifican la validez de la función de recompensa aportada en este proyecto.

Otro experimento importante realizado en este proyecto, trata de evaluar la capacidad de generalización del modelo diseñado. El concepto de generalización se define como la capacidad que tiene el modelo de alcanzar objetivos que no han sido considerados en el entrenamiento, es decir, son nuevos para el modelo. Se realizaron dos experimentos. El primero consiste en alejar los objetivos a evaluar 1, 2, 4 y 8 pasos de los objetivos empleados para entrenar el modelo; y el segundo experimento consistió en incrementar el número de objetivos empleados para el entrenamiento del modelo. Los resultados se pueden observar en la figura 4. Tras la realización y evaluación de este tipo de experimento, se concluye la dificultad de generalizar que presentan los algoritmos basados en *deep* RL.

Tras el diseño y evaluación del modelo propuesto en este TFM se exponen las siguientes conclusiones. Se ha realizado una evaluación exhaustiva del modelo planteado en este proyecto sobre el entorno virtual AI2-THOR, donde los resultados del modelo propuesto reportan los siguientes hechos: 1) mejora el estado del arte en número de pasos y colisiones; 2) es capaz de converger más rápido que un modelo que no tiene en cuenta las



(a) Primer experimento de generalización. (b) Segundo experimento de generalización.

Figura 4: Experimentos de generalización.

colisiones, sino que simplemente busca el camino más corto; 3) se ofrece una interesante capacidad de generalización para alcanzar *objetivos* visuales que no han sido vistos durante el entrenamiento.

Glosario

TFM Trabajo Fin de Máster

RL Reinforcement Learning

IA Inteligencia Artificial

API Application Programming Interface

DA Domain Adaptation

A3C Asynchronous Advantage Actor-Critic

DQN Deep Q-Network

PG Policy Gradient

SLAM Simultaneous Localization And Mapping

ICP Iterative Closest Points

GPS Global Positioning System

NN Neural Networks

CNN Convolutional Neural Networks

MSE Mean Square Error

EBE Entropy-Based Exploration

Capítulo 1

Introducción

Este primer capítulo muestra los tres pilares básicos de cualquier proyecto: motivación para realizarlo, objetivos del mismo, y campos de aplicación en la vida real. Finalmente cerraremos el capítulo con una sección en la que se explica la estructura del presente documento.

1.1. Motivación

Necesitamos que las plataformas robóticas aprendan a navegar de la misma manera que hacemos los humanos: teniendo en cuenta el entorno y los objetos del mismo que nos rodean. Es decir, que de alguna manera aprendan a mantener un espacio de seguridad tanto con los objetos como con las personas que rodean a la plataforma robótica. Especialmente si queremos que la navegación se dote de algún significado, como puede ser por ejemplo ir en búsqueda de un objeto específico localizado en entornos cerrados, (*e.g.* cocinas, dormitorios, salas de estar, etc. Este es el objetivo principal del presente TFM.

El problema de navegación ha tenido un gran impacto en los últimos años. La mayoría de las soluciones para la navegación propuestas han sido consideradas como métodos basados en mapa, *map-based methods*. Este término hace referencia a aquellas soluciones que necesitan de un mapa del entorno en el que se desea navegar de cara a tomar decisiones de navegación dentro del mismo (*e.g.* [4, 5]). Otros métodos reconstruyen el mapa del entorno automáticamente sobre la marcha y lo usan para navegar, *e.g.* [6, 7, 8], o toman directrices basadas en la experiencia humana para la construcción del mapa como ocurre en [9, 10]. En último lugar, cabe resaltar los métodos denominados *map-less*, *e.g.* [2], los cuales no requieren de ningún mapa ya que no tienen ninguna suposición sobre los puntos de referencia de las escenas en las que se pretende navegar. El modelo que presentamos en este TFM sigue el método *map-less*, el cual presenta una solución visual orientada a la navegación que combina los avances más recientes en las tecnologías *deep* y *reinforcement learning*. Así mismo, el desarrollo de este tipo de modelos para la navegación de plataformas robóticas tiene como finalidad la implementación y su funcionamiento en entornos

reales abriendo un amplio abanico de posibles aplicaciones que resuelvan tareas de la vida cotidiana.

La navegación robótica es un sector cuya evolución ha crecido durante los últimos años, saliendo al mercado soluciones aplicables a diversos problemas. Sin embargo, a día de hoy es difícil encontrar soluciones que integren el sector de la navegación de plataformas y el sector de visión por computador. La visión por computador es un sector en continuo crecimiento que plantea resolver problemas de manera más económica. La fusión de ambos sectores, plantea el desafío que se persigue en este TFM, investigando en nuevas tecnologías que permitan resolver de una manera eficiente el problema planteado.

Este desafío presenta una principal ventaja: abaratar el precio del sistema desarrollado. La ventaja más importante de integrar la visión por computador reside en el uso de cámaras cuyo coste está por debajo de algunos de los sensores que se incorporan en las plataformas robóticas, lejos del alcance del consumidor.

Estos hechos promueven el desarrollo del sistema que se expone en el presente proyecto.

1.2. Objetivos

Como ya hemos mencionado en la sección anterior, el objetivo de este TFM consiste en proponer soluciones que promuevan la navegación de plataformas robóticas en las que se adopten comportamientos similares a los que presentamos los humanos. De manera que para lograr este objetivo contestamos a la siguiente pregunta: *¿Cómo aprende a navegar un humano dentro de entornos cerrados?* Para un mejor entendimiento del concepto que se persigue en este proyecto proporcionamos la figura 1.1 ¿Qué trayectoria seguiría un humano para alcanzar la lámpara, la verde o la azul? Creemos que los humanos tienden a navegar evitando posibles colisiones, anticipando las mismas en su trayectoria hacia el objetivo marcado. En otras palabras, nosotros los humanos tratamos de movernos manteniendo un margen de separación con los posibles obstáculos del entorno que nos rodean.

En la figura 1.1 queda de manera esquematizada la idea que se persigue para este TFM. Para tal fin, hicimos uso de la última tecnología para el entrenamiento de plataformas robóticas orientadas a la navegación conocida como *deep* RL [13]. En la figura 1.1, se observan zonas sombreadas de color rojo que representan zonas que el agente (o robot) usará para anticipar las colisiones que se puedan presentar entre el agente y el objetivo asignado. Por otra parte, el entrenamiento de este tipo de algoritmos normalmente se lleva a cabo en entornos virtuales como ya se mencionará en capítulos posteriores, ya que la recopilación de datos en entornos reales puede llegar a ser muy costosa tanto en recursos como en tiempo. Por eso, la filosofía que se plantea en este proyecto de no solo dotar a una plataforma robótica de la inteligencia de navegar hasta un objeto, sino además la capacidad de anticipar posibles colisiones con los objetos interpuestos entre la posición de

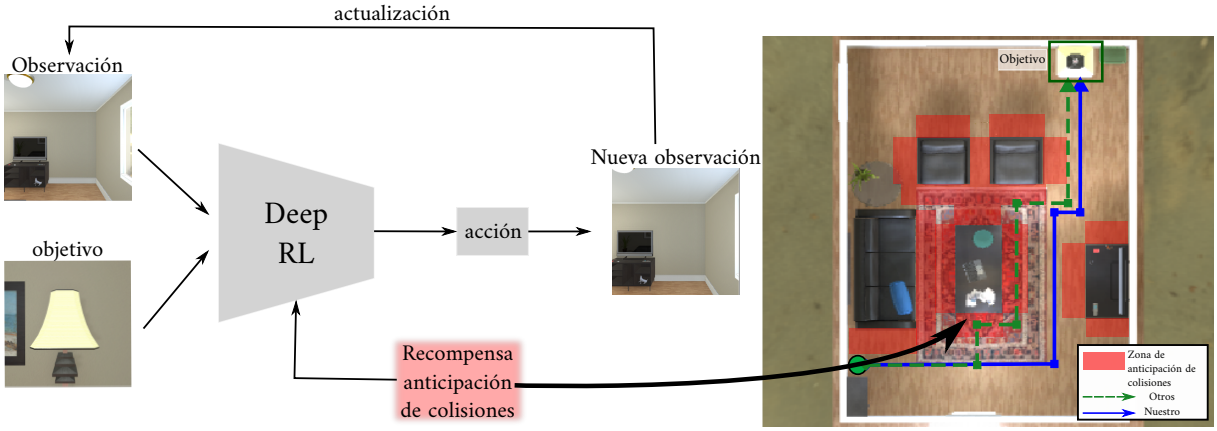


Figura 1.1: ¿Optaría un humano por el camino de color verde o azul para llegar hasta la lámpara? En este proyecto se propone un modelo basado en *deep reinforcement learning* que es diseñado para aprender a navegar hacia un objetivo definido visualmente de la misma manera que un humano lo haría.

la plataforma y su objetivo durante la trayectoria, hace que la solución planteada dote de grandes beneficios a la hora de su funcionamiento en entornos reales.

1.3. Campos de aplicación

Las plataformas robóticas ya juegan un gran papel en nuestras vidas, facilitando tareas de la vida cotidiana, como pueden ser en la limpieza del hogar. Sin embargo, a día de hoy no es tan frecuente encontrar plataformas robóticas cuyo objetivo sea el de la navegación en entornos cerrados, aprendiendo a navegar como hacemos los humanos: evitando colisionar con los objetos del entorno que nos rodean. El sector que se aborda en este TFM, el de navegación de plataformas robóticas, tiene cada vez más un mayor interés para el desarrollo de aplicaciones orientadas a la navegación dentro de entornos de carácter doméstico. Por ello, este TFM busca ampliar soluciones para promover que este tipo de plataformas puedan llegar a integrarse en nuestra vida cotidiana en numerosas tareas, entre las que podemos destacar:

- En este proyecto se aborda únicamente el problema de navegación, pero éste se puede combinar con otros problemas como de interacción con objetos o *grasping*, permitiendo la elaboración de aplicaciones útiles para personas con problemas de movilidad, resolviendo tareas como la de posicionamiento de objetos en algún lugar específico dentro de un hogar, o la recogida de los mismos.

- Como se puede apreciar en el título del presente documento, la solución que se propone está implementada en plataformas virtuales, ya que su uso facilita enormemente la captura de datos para el entrenamiento de este tipo de algoritmos, dado

que necesitan una gran cantidad de estos. Sin embargo, lo utópico de este tipo de soluciones es que interaccionen en entornos reales, resultando una aplicación útil que pueda facilitar, ayudar y apoyar a los humanos.

- Desarrollar soluciones de estimulación sensorial para niños con parálisis cerebral, mediante plataformas robóticas inteligentes en cuanto a tareas de navegación y reconocimiento se refiere. Esta estimulación pasaría por la plataforma de manera que ésta pudiese procesar la información necesaria para seguir a los niños y navegar junto a ellos, de manera que se potenciarían sus capacidades psicomotrices.

1.4. Organización del documento

El presente documento recoge el trabajo realizado para este TFM, que se estructura de la siguiente manera. Este primer capítulo recoge una introducción del objetivo e idea que se quiere resolver en este proyecto. El capítulo 2 recoge la literatura y el marco teórico en el que se basa el trabajo realizado, poniendo en contexto al lector sobre las diferentes líneas de investigación seguidas sobre el tema tratado en este TFM. En el capítulo 3 abarca la definición y explicación del modelo que se presenta en este proyecto. Los experimentos realizados durante la elaboración del TFM con sus respectivos resultados quedan recogidos en el capítulo 4. Finalmente, en el capítulo 5, se recogen las conclusiones realizadas sobre el trabajo realizado, así como las contribuciones de este proyecto y las posibles futuras líneas de trabajo que deja abierto este TFM.

Capítulo 2

Estado del arte

Este capítulo recopila los trabajos más relevantes entorno a la línea de investigación que se ha tratado en este TFM, transmitiendo al lector una visión general del trabajo realizado en el sector. Este capítulo se subdivide en dos secciones. La primera habla sobre trabajos enfocados a la navegación de plataformas robóticas mediante el uso de diversos métodos de navegación. La segunda sección está enfocada a la técnica que se ha empleado en este trabajo para dar una solución de navegación.

2.1. Técnicas tradicionales para la navegación de plataformas robóticas

Desarrollar soluciones robóticas que se comporten de manera autónoma en tareas de la vida cotidiana, es un campo que tiene una larga trayectoria en el área de la robótica. El caso que se abraza en este trabajo, el de la navegación de plataformas robóticas, ha tenido un gran despliegue durante los últimos años con diversas técnicas que han posibilitado desarrollar aplicaciones autónomas. En esta sección se va a hablar del trabajo más relevante en este área, mencionando el tipo de técnica empleada para dar solución al problema de navegación.

En robótica, la navegación es un término que hace referencia a la capacidad de un robot para determinar su propia posición dentro de un *marco de referencia*[1], y acto seguido planificar un camino hacia la localización de un objetivo. El hecho que posibilita a un robot navegar dentro de un entorno, ya sea interior o exterior, es la posesión de una representación, por ejemplo, un mapa del entorno y la habilidad para interpretar esa representación. La navegación puede definirse como la combinación de tres competencias fundamentales: auto-localización, planificación de un camino y construcción de un mapa, e interpretación del mapa. La planificación del camino es un término usado en robótica

[1]En robótica, un marco de referencia, consiste en una abstracción de un sistema de coordenadas y el conjunto de puntos de referencia físicas que colocan y orientan de manera única el sistema de coordenadas y estandariza medidas.

para encontrar una secuencia de configuraciones válidas que mueven al robot del punto de origen al punto de destino marcado. Por ejemplo, un robot movil se considera navegando dentro de un edificio hacia un punto distante. Debe ejecutar la navegación mientras evita las paredes y no caerse por la escaleras. Un algoritmo de planificación del camino tomaría esta descripción de la tarea como entrada, y produce los comandos de velocidad y giro que son enviados a las ruedas. Por otro lado, el término *mapeo robótico* es una disciplina relacionada con la visión por computador. El mapeo robótico es la rama que trata el estudio y la aplicación de la capacidad de localizarse en un mapa o plano, y a veces, de construir el mapa o el plano mediante un robot autónomo. Algunos sistemas de navegación son capaces de localizar y construir un mapa del entorno de manera simultánea. Este técnica se conoce como Simultaneous Localization And Mapping (SLAM) [14], la cual es una de las más usadas en el área de la robótica. Este tipo de técnica ha estado siempre presente en todas las aplicaciones basadas en la navegación de plataformas robóticas. Como por ejemplo en [15], donde se plantea una solución que ataca el problema de auto-localización, cuyo objetivo es la identificación de lugares tras realizar una exploración previa y reconstrucción del mapa de la escena. La solución que se propone en dicho trabajo acumula información sensorial tanto exteroceptiva como propioceptiva para localizar, de forma que no es necesario ningún conocimiento previo para localizar al robot. Otros trabajos han optado por la navegación de plataformas en entornos de exterior. Por ejemplo, en [16], se realiza una combinación del algoritmo Iterative Closest Points (ICP) [17, 18, 19] con la localización basada en la inclinación del terreno resultando posible la reconstrucción de un mapa en un entorno exterior. La información proporcionada por la inclinación del terreno durante la navegación del robot es usada para el cálculo de localización local entre los intervalos de escaneo. En [20] se hace uso de la técnica previamente mencionada, SLAM para dar soluciones de navegación en drones en entornos de denegación Global Positioning System (GPS). El principal objetivo de este trabajo, es dar una solución de bajo coste basado en drones con una cámara monocular para tareas de gestión en almacenes, como pueden ser escaneado del inventariado o la actualización de la posición. Por otro lado, en la línea planteada en este proyecto, la de navegación robótica, se pueden encontrar trabajos que combinan las técnicas para la navegación comentada anteriormente con la visión por computador. Los trabajos [21, 22] son algunos ejemplos. En [21], se plantea una solución para la navegación visual en la que se usa técnicas SLAM con ejecuciones robustas basadas en objetivos en entornos ruidosos. El aprendizaje llevado en este trabajo, permite al modelo aprovecharse de las regularidades estadísticas presentadas en el mundo real. En [22], se expone una solución basada en visión para la navegación de plataformas en entornos de interior capaz de anticipar las colisiones. Para ello, se aborda el problema de la auto-localización mediante un modelo basado en visión que se realiza a través de una correspondencia entre el modelo 3D y la imagen capturada por el robot. En contraste, el trabajo presentado en el presente documento sigue un modelo que no necesita ningún mapa de la escena por la que navega, técnica conocida como *map-less*.

2.2. Soluciones inteligentes para la navegación basadas en *deep* RL

A día de hoy, las disciplinas asociadas al sector de la inteligencia artificial han tomado un papel muy relevante para plantear una solución a cualquier problema planteado o mejora de servicios de la vida cotidiana. En concreto, la navegación de plataformas robóticas autónomas, además de tener gran interés como ya se mencionaba previamente, ha tenido una gran adhesión con estas disciplinas. Como por ejemplo, el *machine learning* y el *deep learning*. El motivo de la integración de este tipo de disciplinas en las plataformas robóticas proviene del propósito de implementar soluciones de bajo coste que puedan estar al alcance de todo el mundo.

En el caso de este TFM, la disciplina basada del sector de inteligencia artificial empleado se conoce como *deep Reinforcement Learning*. Esta disciplina toma como base los principios que sigue el RL. A continuación, se mencionan una serie de trabajos que toman como base esta última disciplina para solucionar un problema planteado. Entre estos, se pueden destacar [23, 24, 25, 26]. En [23], se expone una solución basada en RL en donde se propone una solución real para el problema de despachar ascensores. En [24], se plantea una aplicación que usa técnicas de RL que facilita la posibilidad del vuelo autónomo. Por otra parte, en [25] se diseña un método basado en visión que hace uso del RL para enseñar a un robot a lanzar una bola hacia un objetivo. Finalmente, en [26] se investigan este tipo de técnicas inteligentes para que un robot de cuatro piernas sea capaz de caminar. Abordando el sector que se trata en este trabajo, el de navegación y robótica, se pueden recalcar trabajos como en [27, 28].

La siguiente sub-sección se centra en la evolución sufrida por el *deep Reinforcement Learning*, proporcionando al lector un contexto claro de la trayectoria seguida en dicha disciplina.

2.2.1. Evolución del *deep Reinforcement Learning*

El *deep Reinforcement Learning* es una disciplina que combina los beneficios proporcionados por las Neural Networks (NN) y Convolutional Neural Networks (CNN), que son asociadas a la disciplina conocida como *deep learning*. Ésta última ha sido empleada también para dar solución al problema de navegación que se aborda en este proyecto. Tomando ventaja del alto rendimiento que presentan las *deep* NN, se pueden destacar una serie de trabajos. En [29] usan imágenes monoculares y en [30] imágenes de profundidad para aprender un modelo que evita colisiones usando NN. En [31] se usa la información semántica obtenida de las imágenes gracias a una red profunda, para decidir el comportamiento de un vehículo autónomo, aunque debido a las opciones de movimiento discretas, los comportamientos para la conducción resultaron ser bruscos.

El trabajo presentado en [3] tuvo gran importancia en el desarrollo del *deep* RL. En éste se combinaba el uso de NN profundas con los algoritmos existentes basados en RL

para la estimación de una función que se explicará posteriormente. Esta combinación resultó en un método conocido como DQN, el cual será explicado en detalle a continuación ya que numerosos trabajos que hacen uso de la disciplina del *deep* RL usan dicho método.

Antes de explicar el fundamento del que se compone el método DQN, se va a contextualizar de manera resumida el entorno en el que se encuentra este tipo de redes. Mientras el aprendizaje supervisado trata de aprender a predecir una serie de valores o clases basadas en un etiqueta, el RL trata de cómo un agente debe comportarse dentro de un entorno dado. La manera por la que se enseña a dicho agente a cumplir su objetivo, es mediante la asignación de recompensas o castigos según su rendimiento dentro del escenario. Por tanto, el RL consiste en entrenar un agente que interacciona con su entorno. El agente interacciona con el entorno mediante la ejecución de acciones, recibiendo una representación nueva del entorno cada vez que ejecuta una, conocido como estado. En cada acción que realiza el agente, éste recibe una recompensa, la cual puede ser positiva o negativa. El único propósito del agente es maximizar la cantidad de recompensa coleccionada por el agente a lo largo del episodio, que es todo lo que ocurre desde el estado inicial al estado final. Por tanto, se refuerza al agente a volver a realizar ciertas acciones mediante la asignación de recompensas positivas, y evitar que realice otras acciones mediante la asignación de recompensas negativas. Así es como el agente aprende a desarrollar una estrategia o lo que también se conoce como una política.

Cada estado del agente es una consecuencia directa del estado anterior y de la acción elegida. El estado anterior a su vez es consecuencia de su estado previo, y así hasta que se alcanza el estado inicial. Cada uno de estos pasos, y su orden, contienen información acerca del estado actual y por tanto, tienen un efecto directo en la próxima elección de la acción que debe tomar el agente. Esta manera de proceder tiene un problema obvio, y es que a medida que se van tomando pasos en el entorno, el agente necesita guardar y procesar cada uno de los pasos tomados. Esto puede resultar en un punto en el que sea inviable realizar cálculos. Para afrontar este problema, se asume que todos los estados son estados de Markov. Es decir, cada uno de los estados sólo dependerá del estado anterior y de la transición de dicho estado al actual.

Una vez aclarados estos conceptos, es momento de desarrollar la primera estrategia. Se va a plantear al lector el caso más simple para así asegurar un correcto entendimiento de este tipo de métodos. Se asume que ya se sabe la recompensa esperada para cada acción en cada paso. ¿Cómo se elegiría una acción en este caso? Bien, sencillamente se escogería la acción que eventualmente genere la máxima recompensa. Esta recompensa acumulada se conoce como *Q Value*. Se puede formalizar matemáticamente la estrategia de la siguiente manera:

$$Q(s,a) = r(s,a) + \gamma max_a Q(s',a)\cdot \tag{2.1}$$

La ecuación 2.1 establece que los *Q Value* resultados de estar en el estado s y seleccionar la acción a, son las recompensas *intermedias* recibidas, $r(s,a)$, más el *Q Value* más alto

posible del estado s' (el cual es el estado al que se llega por tomar la acción a en el estado s). Se recibirá el mayor valor de Q para el estado s', eligiendo la acción que maximice el *Q Value*. Además, se introduce el término γ, conocido como factor de descuento. Este se encarga de controlar la importancia de las recompensas a largo plazo en comparación con la inmediata. Dicha ecuación se la conoce como la ecuación de Bellman. Esta ecuación es poderosa y será de gran utilidad debido a dos importantes características:

- Conservando los supuestos de los estados de Markov, la recursiva naturaleza de la ecuación de Bellman posibilita que las recompensas de los estados futuros se propaguen a los estados pasados lejanos.
- No hay necesidad de saber realmente cuáles son los verdaderos valores de Q iniciales. Dada su recursividad, se puede suponer unos valores que con el tiempo convergerán a los valores reales.

Por tanto, ya tenemos una estrategia para determinar qué acción elegir ante un estado dado. Pero, ¿cómo se podría implementar este algoritmo para resolver desafíos en la realidad? Una opción, puede ser dibujando una tabla en la que se almacenen todas las posibles combinaciones de estados y acciones, y guardar ahí los valores de Q, los cuales se actualizarán mediante el uso de la ecuación 2.1. Aun así, este algoritmo plantea un problema. Si el agente escoge siempre las mismas acciones que resulten los valores de recompensa más altos, nunca se probarán cosas nuevas, de manera que el algoritmo podría perder un enfoque que aporte una mayor recompensa por el hecho de nunca haberlo probado. En RL se conoce como el problema de *explotación vs exploración*. Para solucionarlo, se hace uso de lo que se conoce como enfoque $\epsilon - greedy$, en el que para $0 < \epsilon < 1$, se escoge una acción *greedy* con una probabilidad $p = 1 - \epsilon$, o una acción aleatoria con una probabilidad $p = \epsilon$. De esta manera, se permite al agente la oportunidad de explorar nuevas oportunidades.

Este algoritmo se conoce como *Q Learning*, el cual ha sido usado en numerosos problemas de RL. Sin embargo, este algoritmo tiene un problema. Uno puede preguntarse, acerca de la capacidad de escalado que presenta el algoritmo *Q Learning*. Es decir, qué ocurriría si el número de estados y acciones resulta ser muy alto. La naturaleza de DQN proviene de solventar este problema. La idea consiste en aproximar los valores de Q mediante un modelo de machine learning, basado en NN. Dicho modelo se le conoce como *función de aproximación*, y es denotado como $Q(s, a; \theta)$, donde θ representa los parámetros entrenables de la red. Como en toda red, hay que definir una función de coste. En este caso, se hace uso de la ecuación 2.1. En ella, se pretende igualar ambas entidades del símbolo igual, resultando en la ecuación planteada 2.2 mostrada a continuación:

$$Cost = [Q(s, a; \theta) - (r(s, a) + \gamma max_a Q(s', a; \theta))]^2 \cdot \tag{2.2}$$

Esta función de coste puede resultar familiar al Mean Square Error (MSE):

$$MSE = \frac{1}{n} \sum_{1}^{n} (y_i - y_i')^2 \cdot \tag{2.3}$$

En este caso, el actual valor de Q es la predicción y, y los valores futuros de recompensa son el objetivo, y'. Una vez definida la función de coste, se introduce el tipo de arquitectura de método DQN. Siguiendo el esquema del algoritmo *Q-learning* que se introducía previamente, una posible solución para la arquitectura de red es la mostrada en la figura 2.1.

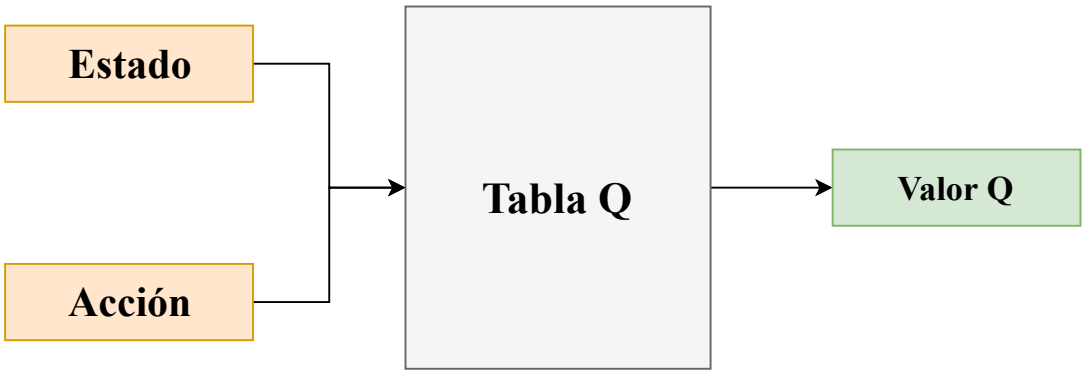


Figura 2.1: Arquitectura para el algoritmo *Q Learning*.

En la figura 2.1 se introduce como entrada tanto el estado como la acción para determinar el valor de Q, así como se establecía previamente. Sin embargo, ahora la función de coste requiere el máximo valor de Q futuro, por lo tanto es necesario que se realicen numerosas predicciones para un solo cálculo del coste, lo cual resulta ser muy ineficiente desde un punto de viste técnico. Por ello, en el algoritmo DQN se propone otra arquitectura de red, la cual es mostrada en la figura 2.2

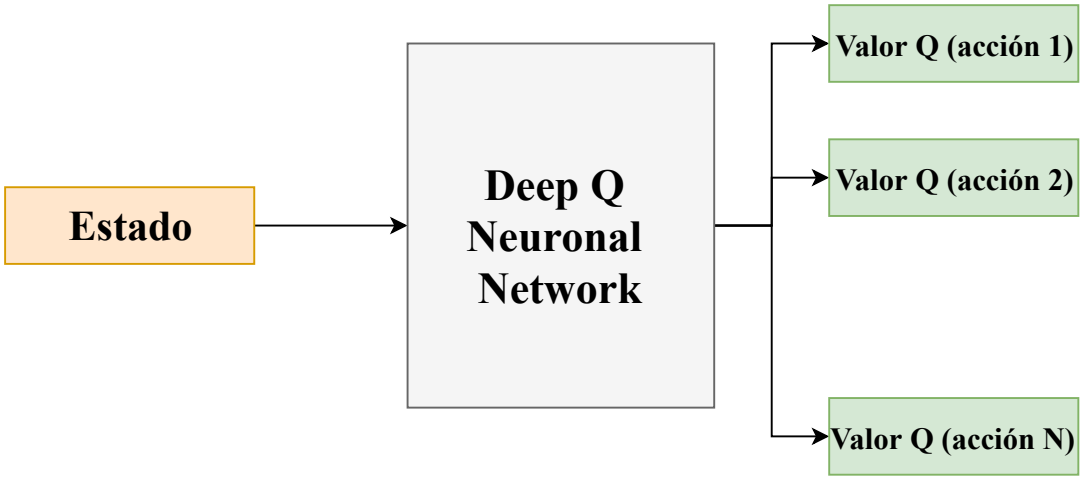


Figura 2.2: Arquitectura para el algoritmo DQN.

En la figura 2.2 se proporciona a la red solamente el estado s como entrada, y se obtiene como salida todos los posibles valores de Q para todas las acciones posibles, haciendo el calculo del coste mucho más eficiente en comparación con la arquitectura presentada en la figura 2.1. En [12] se plantean versiones asíncronas de este algoritmo. Además

se introduce el algoritmo empleado en este trabajo, que será presentado en la sección 3.2.3.

Como ya se ha mencionado previamente, el método DQN ha sido empleado en numerosos trabajos. Para el problema que se plantea resolver en este TFM en concreto, se destacan diversos trabajos. En [32] se hace una comparación entre el método Entropy-Based Exploration (EBE) y el método $\epsilon - greedy$ que se explicaba anteriormente, evaluando la capacidad de un agente para alcanzar un objetivo en entornos simulados complejos. En este trabajo, la evaluación de los agentes entrenados se lleva a cabo en una versión diferente del entorno para evaluar la capacidad de generalización de la política aprendida. En [33] se plantea una solución también para la navegación de un agente en entornos tanto virtuales como reales. Además, dicho trabajo concibe la disciplina del *deep* RL como una herramienta en la que no sólo un agente aprenda a memorizar los *objetivos* para los que se entrena navegar, sino que también sea posible, a partir de ese entrenamiento, razonar como alcanzar objetivos que no hayan sido vistos durante dicha fase. En [2] se plantea el concepto de *target-driven navigation*, es decir, incluir el objetivo de navegación como entrada al modelo, de manera que el re-entrenamiento para nuevos objetivos es más fácil. Este TFM toma como línea base el trabajo que se plantea en [2], sin embargo, en éste modelo no se tienen en cuenta las colisiones durante la fase de entrenamiento, hecho que se reflejará en la sección de resultados del presente documento.

2.2.2. Cambio de dominio: del virtual al real

Esta última sección cierra el capítulo informando al lector acerca de otra línea de investigación puntera en el sector de la navegación de plataformas robóticas. Este TFM trabaja en entornos virtuales como ya se mencionará posteriormente. Uno de los motivos por los que se trabaja con este tipo de entornos, es la inmensidad de datos disponibles para poder entrenar este tipo de algoritmos. A pesar de que las imágenes usadas sean muy realistas, siempre existirá una diferencia entre una imagen tomada de un entorno virtual y un entorno real, este problema es conocido como *reality gap*. El consumo de tiempo y recursos que lleva realizar este tipo de entrenamientos en entornos con imágenes reales, plantea poner una solución que permita adaptar las políticas aprendidas en un entorno virtual a políticas que puedan ser empleadas en entornos reales. Este tipo de técnica es conocida en la literatura como Domain Adaptation (DA). El desarrollo de este tipo de técnicas es de vital importancia de cara a exportar un algoritmo aprendido en un entorno virtual, a una plataforma que interaccione con el mundo real. La técnica DA puede realizarse de tres maneras distintas, así como se cita en la literatura: mediante la adaptación del dominio a nivel de características, conocido en la literatura como *feature-level domain adaptation*; mediante la adaptación del dominio a nivel de pixel, apareciendo en la literatura como *pixel-level domain adaptation*; o mediante la aleatorización del dominio, citado en la literatura como *domain randomization*.

La primera de estas técnicas, *feature-level domain adaptation*, se centra en aprender características de las imágenes invariantes en el dominio, tanto para aprender una transformación fija entre características pre-computadas entre el dominio de origen y el dominio de destino, como para aprender un extractor de características invariante en el dominio, normalmente representado por una CNN. La segunda técnica, *pixel-level domain adaptation*, tiene como objetivo que las imágenes del dominio origen se parezcan lo más posible a las imágenes del dominio fuente, de manera que el aprendizaje en el dominio origen es mucho más efectivo para la evaluación en el dominio destino. Finalmente, el método de aleatorización del dominio consiste en aleatorizar la textura de los objetos, las condiciones de luz, y las posiciones de la cámara durante la fase de entrenamiento, de manera que el modelo aprendido tenga una capacidad más natural de generalizar en escenarios reales.

Hay numerosos trabajos que investiguen en las líneas mencionadas anteriormente. Uno de ellos puede ser [34], el cual sigue el método de adaptación de dominio a nivel de características. El modelo presentado se compone de dos partes: una adaptación de características, y una parte de transferencia de política. El objetivo es que un clasificador mapee las imágenes sintéticas cercanas a las imágenes reales en el espacio de características, de manera que el clasificador no debe diferenciar entre imágenes sintéticas y reales. La segunda parte del modelo consiste en transferir las políticas entrenadas en el dominio virtual para que puedan ser aplicadas en el dominio real. En cuanto a trabajos que investiguen la adaptación de dominio a nivel de pixel, se pueden destacar [35, 36]. En [35] se usa un modelo que emplea imágenes de profundidad. La problemática surge al estimar las imágenes de profundidad a partir de una red alimentada por imágenes RGB, ya que las imágenes de profundidad estimadas son muy ruidosas con respecto a las usadas en el entorno virtual. La solución que se plantea consiste en la adaptación de las imágenes del dominio virtual introduciendo ruido, de manera que son más similares a las imágenes estimadas por la red mencionada anteriormente. Por el contrario, en [36] se actua de manera inversa. En este caso, se pretende que las imágenes del dominio real se parezcan lo máximo posible a las imágenes empleadas para el proceso de entrenamiento. De esta manera, los parámetros aprendidos para las imágenes sintéticas serán útiles al procesar imágenes que sean muy similares a las empleadas para aprender dichos parámetros. Finalmente, en [37] se plantea un modelo que sigue el último método mencionado anteriormente conocido como *domain randomization*. El objetivo de este trabajo consiste en vencer las diferencias entre las imágenes sintéticas y reales, entrenando al modelo en un entorno virtual que aleatoriza la posición de sus objetos, posibilitando al modelo ver una gran cantidad de entornos durante la fase de entrenamiento. Se cierra este capítulo resaltando que este tipo de técnicas no han sido empleadas en el desarrollo del trabajo presentado en este TFM.

Capítulo 3

Solución para la navegación inteligente

Este capítulo alberga el contenido necesario para comprender el modelo que se propone en este TFM para la navegación de plataformas robóticas en entornos virtuales. Se expone el entorno virtual empleado para entrenar la plataforma robótica, así como la extracción de los datos necesarios para dicho fin. Además, se presenta el modelo de navegación basado en *deep* RL, detallando su funcionamiento y los algoritmos que lo forman.

3.1. Entorno virtual y base de datos

El trabajo recogido en este documento presenta un modelo basado en técnicas de *deep* RL. Este tipo de modelos requieren una gran cantidad de datos para que los algoritmos puedan converger a una solución óptima. Esta recogida de datos en entornos reales, conlleva no solo un gran gasto de recursos (pues sería necesario dotar de una plataforma en la que poder recoger los datos), sino también de tiempo, en donde el error humano puede jugar un papel importante. Por otro lado, el entrenamiento de este tipo de algoritmos en entornos reales, en donde las políticas de movimiento no han convergido a una solución óptima, puede generar grandes daños a la plataforma, suponiendo un coste extra. Por este motivo se empezó a desarrollar entornos virtuales, donde se dispone de un agente[1]. Al disponer de un entorno virtual, la cantidad de datos que se disponen aumenta de manera exponencial, facilitando el entrenamiento de este tipo de algoritmos.

En este TFM se ha optado por el uso del entorno virtual conocido como *AI2-THOR* [1]. Este nombre proviene de *The House Of inteRactions*, liberando un *framework* para la investigación de Inteligencia Artificial (IA) visual. AI2-THOR dispone de escenas de interior en 3D con imágenes muy realistas, en donde un agente de IA puede navegar y desempeñar numerosas tareas. Las escenas que abarca este entorno se clasifican en cuatro categorías dentro del entorno doméstico: cocina, baño, sala de estar y dormitorio. Una de las características por las que destaca este entorno, es la capacidad que se le transfiere

[1] El término agente dentro de un entorno virtual equivale a una plataforma robótica. Es decir, el agente es el que toma las decisiones dentro de un entorno virtual.

al agente para poder interactuar con los objetos, cambiando así el estado de la escena. Dentro de las escenas, el agente puede encontrar objetos como sillas, las cuales puede empujar, o armarios y frigoríficos que el agente es capaz de abrir.

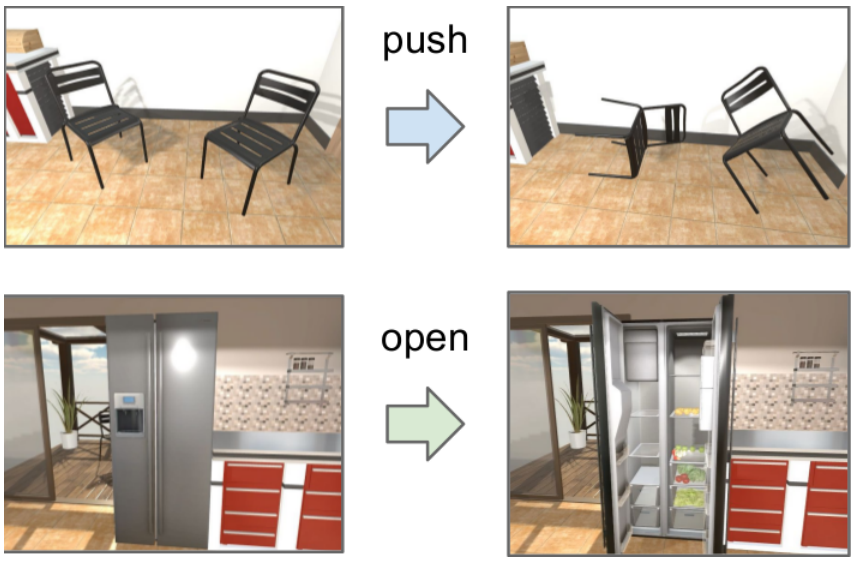


Figura 3.1: Ejemplo del entorno virtual AI2-THOR. Fuente: [1]

En la figura 3.1 se puede apreciar como el agente es capaz de alterar el estado de algunos objetos, permitiendo desarrollar soluciones que no solo naveguen en las escenas, sino que también puedan interaccionar con los objetos, conocido como *grasping*. Además este *framework* cuenta con una Application Programming Interface (API) de *Python*, permitiendo de forma sencilla el control del agente dentro del entorno. Más adelante, en esta sección, se comentará cómo se empleó esta herramienta para la extracción de datos.

Como ya se mencionó anteriormente, este TFM toma como línea base el trabajo presentado en [2]. Por ello, se empleó la misma base de datos que en [2], realizando un análisis comparativo con dicho modelo. A continuación, se va a exponer la estructura de datos empleada en este proyecto.

La solución de navegación que se propone en este TFM al igual que en [2], se define como navegación discreta. Este tipo de navegación consiste en ejecutar acciones discretas dentro de un espacio de movimiento. Por ello, se puede pensar en una escena como una *rejilla*, la cual esta compuesta de una serie de celdas. Cada una de estas celdas, se compone de cuatro puntos de vista diferentes, siendo cada uno de ellos etiquetados con un identificador. El agente tendrá en todo momento asociado un identificador, el cual proporciona una serie de información. Es necesario hacer esta aclaración, de manera que al lector le quede claro el proceso de *renderización*[2] llevado a cabo para conformar la base de datos usada en este proyecto.

[2]El término *renderización* hace referencia a la generación de imágenes visibles e inteligibles para el ser humano a partir de información digital (en este caso la proporcionada por el entorno virtual).

La estructura de datos empleada por el modelo que se presenta en este trabajo contiene los siguientes campos:

- **Localizaciones.** Este campo indica cada una de las posiciones (expresadas en coordenadas (x, z), de acuerdo con el entorno virtual) que el agente toma dentro de una escena.
- **Rotaciones.** Este campo indica la orientación del agente dentro de una de las celdas en la escena. Esta orientación puede adoptar cuatro valores distintos: 0, 90, 180 o 270. En cada celda el agente dispondrá una observación por cada una de estas rotaciones.
- **Observaciones.** Este campo recoge todas las imágenes posibles dentro de una escena. De esta manera es posible reproducir lo que el agente ve dentro de una escena.
- **Gráfico de transición.** Este campo es de vital importancia para la navegación discreta que se plantea en este TFM. Consiste en una matriz con un número de filas igual al número total de puntos de vista, es decir, de identificadores, y cuatro columnas. Cada una de esas columnas simula los cuatro diferentes movimientos que se pueden realizar dentro del entorno: moverse hacia delante, rotar a la derecha, rotar a la izquierda y moverse hacia atrás. Cada vez que se realiza un movimiento, el entorno devuelve un nuevo identificador. Este nuevo identificador se obtiene comprobando en el gráfico de transición la columna asociada a la acción tomada.
- **Características.** Este campo recoge un vector de características para cada una de las entradas del campo *observaciones*. Este vector se obtiene al pasar cada una de las imágenes por el modelo de red ResNet-50 pre entrenado en la base de datos ImageNet. Se trunca el modelo en la última capa antes del clasificador, resultando un vector de longitud: 2048. Este vector es la entrada al modelo de navegación que se presenta en este trabajo.

A modo esquemático, representando los campos explicados en el párrafo anterior, se adjunta la figura 3.2. En ella, se aprecia el concepto de *rejilla* mencionado, que da lugar a la navegación discreta que se plantea en este TFM En la figura 3.2 se puede observar el agente dentro de una escena, por medio de una vista desde arriba de la escena. En la parte derecha de la figura, se pueden observar las cuatro posibles vistas que se pueden obtener dentro de la celda (siendo igual para el resto de celdas que componen la escena). Como se puede apreciar, cada observación tiene un *ID* asociado, el cual es único en toda la escena, es decir, no hay dos puntos de vista con el mismo *ID*. En la parte inferior, se representa un pseudo ejemplo de gráfico de transición para el estado: $ID = 15$ (estado en el que se encuentra el agente en este ejemplo). Este gráfico se interpreta de la siguiente manera: si el agente decide moverse hacia delante, pasaría a estar en el *ID:* 25; en caso de rotar hacia la derecha o hacia la izquierda, el agente se situaría en el *ID* 16 o 18, respectivamente.

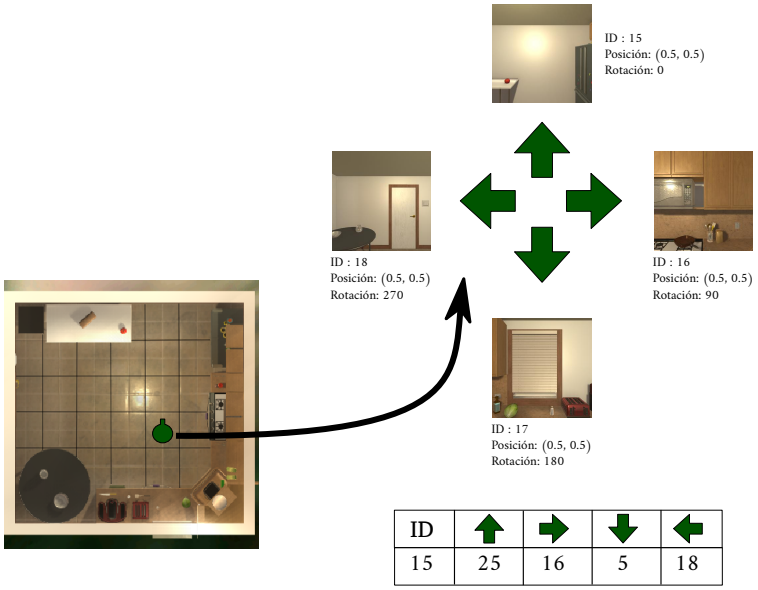


Figura 3.2: Proceso de renderización en el entorno virtual AI2-THOR.

Finalmente, en caso de ejecutarse la acción *moverse hacia atrás*, el agente alcanzaría el *ID:* 5. En caso de que no se pueda ejecutar un movimiento, por ejemplo, hay un objeto o una pared, se anota en el gráfico de transición con el valor -1, de esta manera se tiene un control de las colisiones producidas durante la navegación. Este mismo ejemplo, se aplica al resto de puntos de vista de la escena, resultando así el campo *gráfico de transición* de la base de datos. Los campos mencionados y explicados anteriormente que componen la base de datos empleada en este proyecto, mantienen coherencia entre sí, por medio de estos identificadores.

3.2. Modelo de navegación con anticipación de colisiones

Esta sección presenta el modelo para la navegación en el que se basa este TFM. A continuación se expone el problema que pretende resolver este modelo. Una vez definido el problema, se muestra el modelo implementado, aportando al lector los detalles necesarios para un claro entendimiento del mismo, haciendo énfasis en los algoritmos y modelos de red empleados.

3.2.1. Descripción del problema

El problema que se plantea resolver se mencionó brevemente en la sección 1.2, al exponer los objetivos marcados para el proyecto.

Este trabajo pretende resolver el problema que se conoce como navegación visual. Este

tipo de navegación, como bien su nombre indica, implica el movimiento de una plataforma robótica a partir de imágenes, o como es en el caso de este proyecto, observaciones capturadas por la plataforma durante la navegación.
La solución de navegación que se plantea en este trabajo, sigue la hipótesis de que las personas nos movemos, en un entorno cerrado, manteniendo un margen de separación entre los objetos que nos rodean. Por tanto el problema planteado consiste en dotar a la plataforma robótica, la inteligencia necesaria para navegar hasta la posición en la que fue tomada la imagen especificada como *objetivo*, de manera que el número de colisiones sea mínimo, como se mostraba anteriormente en la figura 1.1.

En las secciones siguientes, se expone el modelo que permite resolver el problema previamente expuesto mediante una técnica conocida como *deep* RL.

3.2.2. La navegación basada en *deep* RL

Este TFM, implementa una solución para la navegación visual basada en la tecnología conocida como *deep* RL. Para familiarizar al lector con la terminología usada en RL, y así asegurar un correcto entendimiento del modelo planteado en este proyecto, esta sección explica de manera breve en qué consiste la navegación basada en RL.

La tecnología conocida como *Reinforcement Learning*, consiste en aprender cómo actuar en función de la situación planteada, de manera que una señal de recompensa (conocida y normalmente mencionada como *recompensa*) sea máxima. El aprendiz (conocido también como agente) sabe las acciones que puede tomar dentro del entorno, pero no cuales ha de tomar acorde a la situación, de manera que la señal de *recompensa* se maximice. En el momento en el que el agente es capaz de tomar las decisiones que en cada instante de tiempo resulten la máxima señal de *recompensa*, se entiende que se ha aprendido correctamente una *política*.

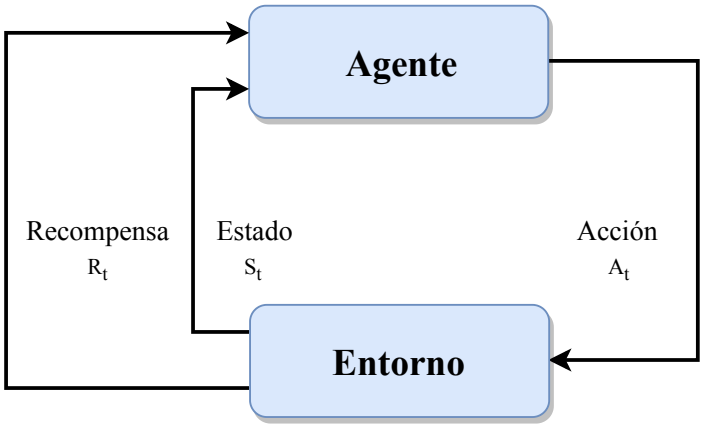


Figura 3.3: Esquema de funcionamiento de los modelos basados en *deep* RL.

La figura 3.3 recoge de manera esquemática el funcionamiento de un modelo basado en RL. El funcionamiento es el siguiente: el agente ejecuta una acción en el entorno. El

entorno ante esta acción devuelve al agente un estado y un valor de recompensa. En función de esos valores, el agente vuelve a ejecutar una acción. Este proceso se repite hasta que la política de movimiento que se desea aprender ha convergido.

Una vez mencionados los aspectos básicos que componen un modelo basado en RL, es momento de exponer los componentes clave que forman parte del modelo presentado en este proyecto. Como se puede observar en la figura 3.3, los componentes principales son: acciones, estados y recompensas. A continuación se hablará de ellos para el problema que se desea resolver en este TFM.

- **Espacio de acciones.** Como bien se ha mencionado varias veces a lo largo de este documento, el objetivo del modelo de navegación que se presenta en este proyecto consiste en mapear una entrada de alta dimensionalidad en 2D (en este caso una imagen capturada por el agente) a una acción dentro del espacio tridimensional (en este caso el entorno virtual). Todos los entornos usados en este trabajo han sido discretizados como se mencionaba en la sección 3.1, de manera que el espacio por donde se pretende navegar queda dividido en celdas. Para cada celda, el agente es capaz de tomar cuatro acciones diferentes: moverse hacia delante, rotar a la derecha, rotar a la izquierda y moverse hacia atrás.

- **Observaciones.** El agente está equipado con una cámara en primera persona que capta las observaciones de la plataforma. Cabe resaltar que cada una de las imágenes se corresponde con cada uno de los estados s_t en un instante de tiempo t entre todos los estados posibles $\mathcal{S}$, que devuelve el entorno virtual ante la ejecución de una acción por parte del agente. El agente debe navegar hacia la posición dentro del entorno, en donde la imagen objetivo es tomada. En este caso, las imágenes usadas, son imágenes RGB de 300×400.

- **Función de recompensa.** La señal de recompensa define el objetivo del problema basado en RL que se pretende resolver. En cada instante de tiempo, el agente ejecuta una acción y el entorno devuelve un número conocido como *recompensa* r_t en el instante t. El único objetivo del agente consiste maximizar el valor de la recompensa total a largo plazo. Por tanto, la señal de recompensa define qué eventos o situaciones son buenas o malas para el agente. A modo de ejemplo, en sistemas biológicos podemos pensar en términos de *recompensa* como el placer o dolor que se experimenta al aprender algo nuevo. La señal de recompensa es la base principal para alterar la *política*; si se escoge una acción de acuerdo a una política, y trae un valor de *recompensa* bajo, entonces se cambia la política para que cuando se vuelva a repetir esa situación se ejecute una acción que traiga asociada un valor de *recompensa* más alto. En este TFM se propone una función de *recompensa* para la anticipación de colisiones, la cual permite al agente aprender a navegar dentro del entorno virtual anticipando los posibles obstáculos que se encuentre. Antes de introducir la función diseñada en este trabajo, se definen los cuatro estados que hay que

tener en cuenta. El primero de los estados se denomina s_{step}, el cual se corresponde a cada vez que el agente realice una acción dentro del entorno de navegación, es decir se mueve a una posición dentro de la *rejilla* siempre y cuando se consideren las siguientes excepciones. Si el agente llega a su objetivo, el estado es considerado como s_{terminal}. Si el agente se localiza en una celda en la que haya un objeto o una pared que pueda provocar una colisión al desplazarse hacia delante o hacia detrás, se nombrará a ese estado como s_{colision}. En último lugar si el agente se encuentra en una celda previa a éste último estado, es decir, dos celdas antes de que se pueda ocasionar una colisión durante la navegación, en ese caso el estado se denomina como $s_{\text{colision}-1}$. Como uno puede apreciar, estos dos últimos estados tienen como objetivo dentro de la función de recompensa anticipar las posibles colisiones que puedan surgir durante la navegación. La función de recompensa que se diseña en este trabajo debe tener en cuenta las colisiones. Por ello, se devuelven unos valores numéricos de *recompensa* diferentes en función del nuevo estado inducido por la acción tomada dentro del entorno en un instante de tiempo determinado, es decir, s_t.

$$f(s_t) = \begin{cases} 10 & \text{if} \quad s_t = s_{\text{terminal}} \\ \alpha & \text{if} \quad s_t = s_{\text{step}} \\ \beta & \text{if} \quad s_t = s_{\text{colision}} \\ \gamma & \text{if} \quad s_t = s_{\text{colision}-1} \end{cases} . \tag{3.1}$$

El estado terminal recibe el máximo valor de *reward*, indicando al algoritmo que el agente ha alcanzado su objetivo. Para el resto de estados se asigna un valor pequeño y negativo, siendo: $\alpha = -0{,}01$, $\beta = -0{,}02$ and $\gamma = -0{,}011$. Se resalta al lector el hecho de que los valores asignados para los estados de anticipación de colisiones son más grandes que el asignado para la fase de movimiento (s_{step}). La intención que se pretende conseguir, consiste en que el agente aprenda a navegar alejado de los estados s_{colision} y $s_{\text{colision}-1}$. Expresado en otras palabras, el agente aprende a anticipar los estados en los que se pueda producir una colisión, resultando una navegación más segura.

La figura 3.4 ejemplifica la función de recompensa explicada anteriormente. En esta figura, se ha dividido la escena en celdas, así como ya se ha explicado en secciones anteriores. Cada celda ha sido coloreada de un color, como se puede observar en la leyenda de la figura 3.4. De esta manera, se consigue reducir el número de colisiones sufridas durante la navegación, por tanto, éstas son capaces de anticiparse. El hecho de que se diseñe una función de recompensa que penalice la navegación por los estados s_{colision} y $s_{\text{colision}-1}$, no implica que no se pueda navegar a través de estos estados. Como ya se mencionó previamente, el objetivo del agente consiste en maximizar la función de recompensa a largo plazo, es decir el modelo basado en *deep* RL usa recompensas acumulativas. Esto permite al agente tomar peores decisiones (es decir, una acción que tenga asociada un *reward*

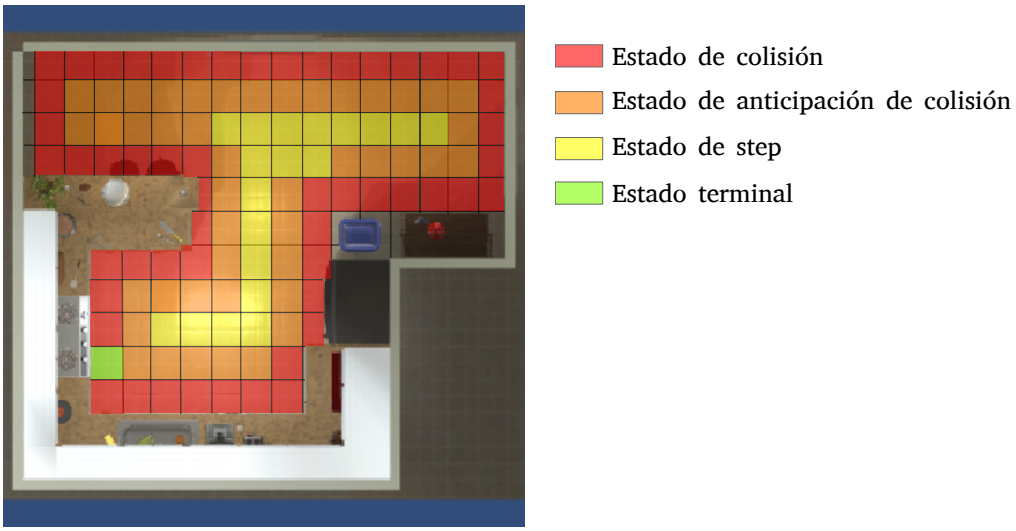


Figura 3.4: Ejemplo de la función de recompensa para la anticipación de colisiones.

negativo) en ciertos momentos del episodio[3], así al final se obtiene el máximo valor de recompensa. Por ejemplo, si el agente se encuentra dos celdas por encima del *objetivo* en la figura 3.4, obtendrá una mayor recompensa si directamente atraviesa esos estados de colisión, frente a dar un rodeo buscando estados de anticipación de colisión o estados de *step* en cuyo caso el valor de *reward* obtenido al llegar al objetivo será mucho menor. De esta manera el número de colisiones se reduce como se mostrará en la siguiente sección.

De acuerdo con el esquema que describe el problema de RL que se desea resolver, se puede resumir esta sección de la siguiente manera. En cada ejecución de una acción en el instante de tiempo t, el agente recibe una representación del estado del entorno, es decir $s_t \in \mathcal{S}$. Basándose en esta información, el agente selecciona una acción, $a_t \in \mathcal{A}(s_t)$, donde $\mathcal{A}(s_t)$ representa todas las acciones posibles para el estado s_t. Como consecuencia de la acción tomada, el agente recibe una recompensa, $r_t \in \mathcal{R}$ y descubre un nuevo estado s_{t+1}. La forma en la que el agente selecciona una acción de acuerdo al estado en el que se encuentre, queda recogida bajo una *política* de aprendizaje, la cual se denota por π_t. Por tanto, $\pi_t(s_t, a_t)$ indica la probabilidad de que el agente seleccione la acción a_t, estando en el estado s_t. El agente cambia la política a través de un método de RL, de manera que el agente consigue la mejor cantidad de recompensa en cada episodio de entrenamiento.

[3]Se define episodio como el conjunto de acciones tomadas por el agente hasta que alcanza el objetivo indicado. Se puede fijar un valor umbral, de manera que los episodios no tengan una duración infinita.

3.2.3. Arquitectura y algoritmo de navegación

Esta sección define la arquitectura y el algoritmo empleado para que el modelo aprenda a navegar en entornos virtuales, de manera que sea capaz de anticipar las colisiones, minimizándolas durante la navegación.

El problema que se plantea resolver en este proyecto, se conoce como navegación visual. Este motivo implica que el modelo únicamente recibe entradas visuales, es decir, imágenes. A continuación, en la figura 3.5 se presenta la arquitectura del modelo de red empleado en este proyecto.

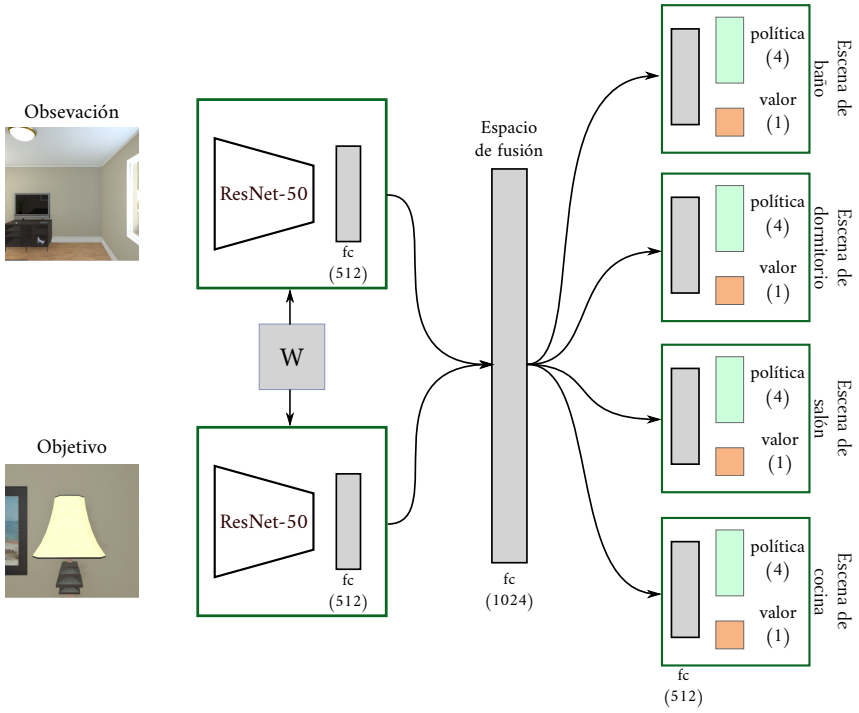


Figura 3.5: Modelo de arquitectura de red presentado en [2]. Las capas siamesas son alimentadas con vectores de características obtenidos del modelo de red ResNet-50 pre entrenado en la base de datos ImageNet. Los parámetros de la ResNet-50 son *congelados* durante la fase de entrenamiento. Cada uno de los agentes (que entrena un *objetivo*) actualiza los pesos compartidos en la red global. En el lado derecho de la figura, se contemplan las redes específicas para cada escena, que dan como salida los valores de la política aprendida según la entrada visual para cada escena en particular.

La figura 3.5 representa el modelo de red que se ha empleado en este TFM. La arquitectura se compone de dos partes: una red global (extendida hasta la capa etiquetada como espacio de fusión) y una red específica (situadas a la derecha de la imagen con el nombre de cada escena al lado).

A continuación, se hará un análisis de la red mostrada previamente, indicando sus partes así como sus capas y su finalidad en el modelo usado en este proyecto.

Lo primero en lo que uno observa en el modelo es su entrada. Se puede apreciar que ésta

se compone de dos imágenes, de las cuales una de ellas es el *objetivo* al que se desea navegar. Esto resalta el enfoque de los autores del modelo en [2], para razonar la disposición entre la ubicación actual y el objetivo proyectando ambas imágenes en el mismo espacio empotrado, donde prevalecen las relaciones geométricas. Las redes siamesas son un tipo de redes neuronales de dos flujos para el aprendizaje de inclusión discriminativo. Por ello, en [2] se propone el uso de dos flujos de capas siamesas de pesos compartidos para transformar el estado actual y el estado del objetivo en el mismo espacio empotrado. Cabe resaltar que la entrada a las capas siamesas consiste en un vector de características de dimensionalidad 2048. Este vector es obtenido de truncar la última capa del modelo de red ResNet-50 (antes de la capa *softmax*), entrenada en la base de datos ImageNet. Dicho vector se calcula para el objetivo y la observación capturada por el agente. La información obtenida de cada uno de estos flujos es fusionada, formando una representación en conjunto que es pasada a las denominadas *capas específicas*, con la intención de capturar información específica para cada una de las escenas, por ejemplo el diseño de la habitación o la colocación de los objetos, cruciales para la tarea de navegación. Las capas específicas se componen de tres componentes. Una primera capa que obtiene información específica para esa escena, y dos capas que obtienen la política de movimiento y el valor. La política consistirá en un vector de cuatro posiciones en el que cada una indica la probabilidad de ejecutar una acción, escogiendo siempre la acción con mayor probabilidad. La salida de la capa valor es empleado durante el proceso de optimización mediante el algoritmo que será explicado a continuación.

Esta separación en dos capas permite un aprendizaje mayor y más rápido del problema que se pretende resolver. Dicho fenómeno es posible gracias al modelo de RL empleado para el proceso de optimización planteado en este trabajo. Este modelo se conoce como A3C [12]. A continuación se hablará de este algoritmo, detallando su origen y eficacia a la hora de resolver este tipo de problemas basados en *deep* RL.

La mejor manera de explicar este algoritmo es desempaquetando su nombre: *asynchronous*, *advantage* y *actor-critic*.

Asynchronous. A diferencia de otros algoritmos para resolver problemas de RL como es el caso de DQN, donde un solo agente es representado por una única red neuronal e interacciona con un único entorno, A3C emplea múltiples modelos simultáneamente para aprender de manera más eficaz. En A3C hay una red global, y múltiples agentes, cada uno de los cuales tiene su propio conjunto de parámetros. Cada uno de estos agentes interacciona con su propia copia del entorno al mismo tiempo que los otros agentes interaccionan con sus respectivas copias del entorno también. La razón por la que este esquema funciona de manera más eficaz que tener un único agente, viene de que la experiencia de cada agente es independiente a la experiencia obtenida por el resto de los agentes durante el proceso de aprendizaje. De esta manera, la experiencia obtenida por todos los agentes durante el entrenamiento, residente en la red global, resulta más diversa.

Actor-Critic. Los modelos de RL se pueden clasificar en dos grandes conjuntos: los mé-

todos basados en el valor[4], y los métodos basados en una política[5]. El algoritmo A3C fusiona las mejores partes de ambos métodos mencionados previamente. Es decir, el algoritmo predice tanto la función de valor $V(s)$, como la función de política óptima $\pi(s)$. El agente usa el valor de la función de valor (denominado según la literatura como *Critic*) para actualizar los parámetros de la función de política óptima (denominado en la literatura como *Actor)*. Análogamente se podría comparar como jugar a un video juego, del que desconocemos sus reglas con un amigo. Mientras uno juega, el amigo va dando su *feedback* acerca de la jugada. Este *feedback* nos ayuda a jugar mejor cada partida. Además, al mismo tiempo que el que juega mejora en el juego, el amigo que da su opinión sobre cada jugada aprende a dar cada vez mejor *feedback*. De esta manera, se aprende mucho más rápido a dominar el juego.

Cabe resaltar que la función de política representa la probabilidad de distribución del espacio de acciones. De manera más técnica y precisa, el agente que aprende a resolver el problema, calcula la siguiente probabilidad condicional: $P(a|s;\theta)$, es decir, la probabilidad parametrizada de que el agente elija la acción a cuando está en el estado s.

Advantage. En uno de los algoritmos que componen el conjunto de métodos basados en un política, conocido como Policy Gradient (PG), se usa para la actualización el retorno con descuento[6] para indicar al agente cuales de las acciones tomadas durante un episodio fueron *buenas* y cuales fueron *malas*. En este algoritmo, la red se actualiza de manera que sea más propenso escoger las acciones que han sido consideradas como *buenas*.

En el algoritmo A3C, la idea de usar estimaciones de lo que se conoce como ventaja, consiste en no solo indicar al agente como de buenas fueron las acciones tomadas, sino como de mejores resultaron ser respecto a lo esperado. De manera intuitiva, esto permite al algoritmo centrarse en las predicciones que la red no llegó nunca a realizar.

El esquema mostrado en la figura 3.5 sigue el algoritmo explicado anteriormente.

[4]Los métodos basados en valor consisten en aprender una función conocida como función de valor, la cual aprende a asignar cada par estado-acción a un valor. Este tipo de métodos es eficiente cuando el espacio de acciones es finito.

[5]Los métodos basados en una política consisten en optimizar una política, sin necesidad de usar una función de valor. Este tipo de métodos es eficiente cuando el espacio de acciones es continuo o estocástico.

[6]En RL se conoce como el retorno con descuento como la suma consecutivas de *recompensas*, de forma que las *recompensas* más distantes son multiplicadas por un factor de descuento elevado a la potencia del número de pasos respecto al instante de tiempo t en el que comenzó el episodio.

Capítulo 4

Resultados

En este apartado vamos a mostrar los experimentos realizados durante la fase de evaluación del modelo que proponemos en este TFM. Los experimentos realizados muestran una comparativa entre nuestro modelo y el modelo presentado por Zhu *et al.* [2].

4.1. Configuración de los experimentos

Todos los experimentos realizados en la línea de investigación que exponemos mediante este TFM tienen dos factores en común: La base de datos y la métrica. Para una mejor comprensión al lector de los resultados obtenidos durante la evaluación del modelo que proponemos, es necesario hacer incisión en una breve explicación de dichos componentes.

Base de datos. Como ya hemos mencionado en secciones anteriores, los datos para entrenar y validar nuestro modelo fueron obtenidos del *framework* AI2-THOR. Este entorno virtual es una fuente de datos excelente para el objetivo de navegación que proponemos en este proyecto, ya que provee imágenes con un nivel de realismo muy elevado de entornos de interior por donde nuestro agente pretende navegar. Técnicamente, seguimos la misma configuración que es liberada en [2]. En este trabajo se proporciona una base de datos obtenida de este entorno virtual, la cual se compone de cuatro diferentes escenas, una por cada categoría: baño, cocina, sala de estar y dormitorio. Todos estos datos están recogidos en un tipo de archivo conocido como *HDF5*, fácil de acceder y estructurado de manera organizada. El tamaño de dichos ficheros puede variar de acuerdo al tipo de categoría almacenada, ya que depende principalmente del número de vistas disponibles dentro del escenario. Por ejemplo, para la escena del salón o del dormitorio el número de imágenes es mayor que para escenas como el baño. Por tanto, el tamaño de este archivo o tamaño de los grafos de transición será mucho mayor.

Métrica de evaluación. Para evaluar la eficacia del modelo de navegación que proponemos en este TFM, usamos, al igual que en [2]: 1) el número de pasos que necesita el agente para alcanzar el objetivo de navegación marcado; y 2) el número de colisiones durante la navegación. Para ambas métricas contra menor sea su valor, mejor es el ren-

dimiento de nuestro algoritmo. También se propone un experimento de generalización en el cual se evalúa la capacidad del modelo de alcanzar objetivos que no han sido vistas durante la fase de entrenamiento. Para este experimento usamos como métrica lo que denominamos como *puntuación de éxito* (*pe*). Consiste en una suerte de puntuación que puede definirse de la siguiente manera. Dada una escena y un objetivo para esa escena, con un número de pasos fijados previamente, calculamos el *pe* como el número de episodios en los que el agente es capaz de llegar al objetivo indicado en un número de pasos que está por debajo de los 500 pasos. Dado que para cada objetivo analizamos el modelo cien veces, el resultante *pe* mide el porcentaje de veces que el agente es capaz de alcanzar su objetivo por debajo del umbral fijo de 500 pasos.

4.2. Resultados en la navegación

4.2.1. Experimento de navegación

En este primer experimento se compara el modelo de anticipación de colisiones que proponemos en este proyecto con el modelo de navegación que tiene el estado del arte de Zhu *et al.* [2]. Hay que destacar que en [2] no se tienen en cuenta las colisiones durante el proceso de aprendizaje, optimizando el modelo para que el agente encuentre únicamente el camino más corto hasta su objetivo. En la tabla 4.1 se puede observar los principales resultados obtenidos para este experimento.

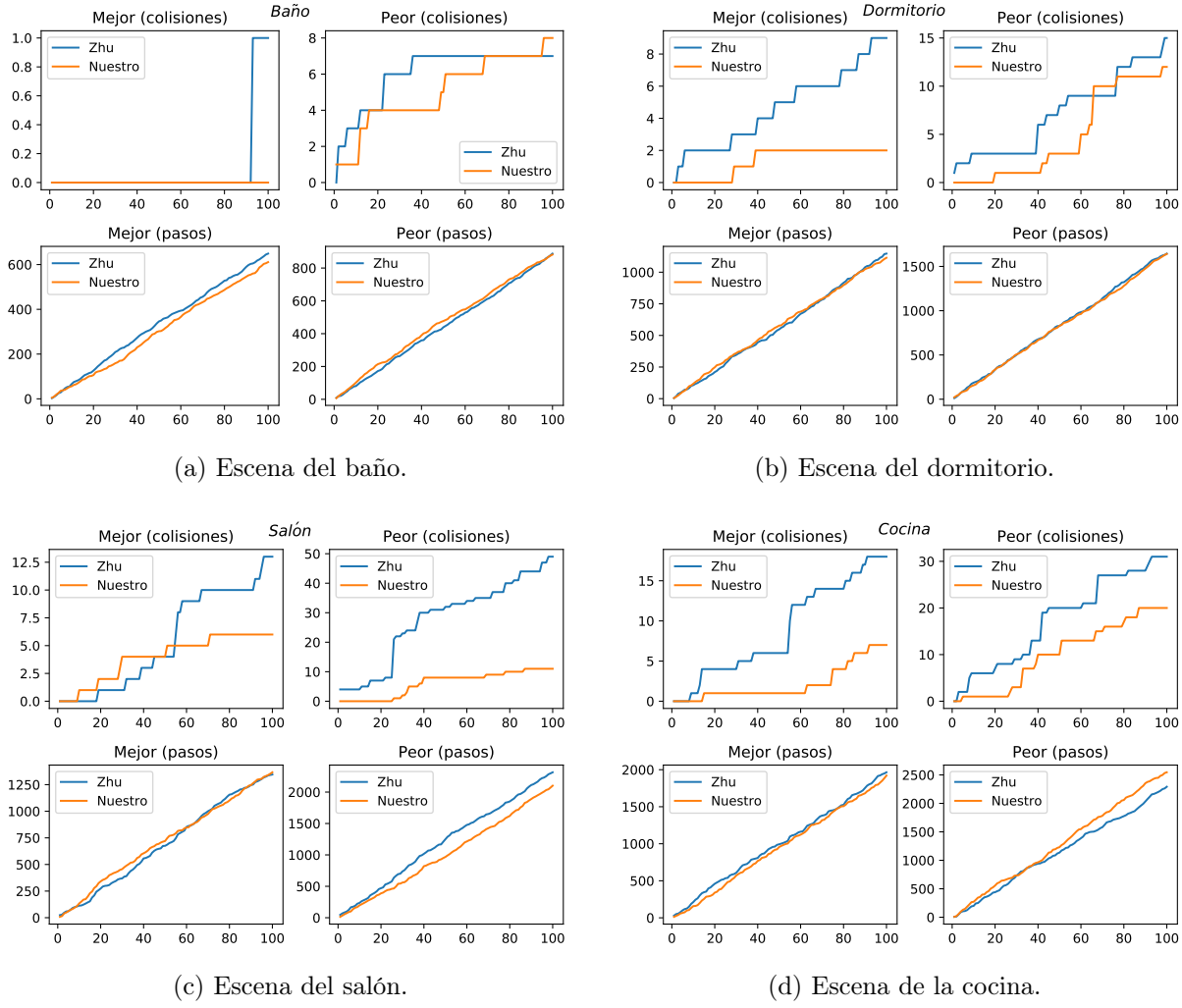
En primer lugar, uno puede observar que el número de colisiones disminuye considerablemente (un orden de magnitud con respecto el otro modelo) siguiendo el modelo que se propone en este TFM. Este hecho reafirma la validez de la función de recompensa propuesta en la sección anterior, que es integrada en el algoritmo de RL propuesto. Además, cabe resaltar que no solo se disminuye el número de colisiones que el agente sufre durante los episodios realizados, sino que también se disminuyen el número de pasos realizados por el agente. En otras palabras, el modelo propuesto en este trabajo es capaz de buscar de manera conjunta la trayectoria más corta entre la posición del agente y su objetivo, anticipando las posibles colisiones que puedan ocasionarse durante dicha trayectoria.

	Escena	Baño	Dormitorio	Salón	Cocina	Promedio
Zhu [2]	Pasos	7.17	14.82	15.2	21.38	14.7
	Colisiones	0.04	0.12	0.25	0.2	0.15
Nuestro	Pasos	7.33	14.81	14.9	20.83	**14.47**
	Colisiones	0.03	0.04	0.15	0.12	**0.082**

Tabla 4.1: Resultados para el experimento de navegación.

Para respaldar los datos mostrados en la tabla 4.1, en la figura 4.1 se refleja el número de colisiones y de pasos acumulados para 100 episodios de navegación, es decir, que el número de pasos o colisiones reflejados en la gráfica para el segundo episodio se compondrá

del número de pasos o colisiones del primer episodio más los que se realicen en el segundo episodio. Así de manera sucesiva hasta completar los 100 episodios. Para ambos métodos (el propuesto en este TFM y el de Zhu *et al.* [2]), se muestra el mejor y el peor de los casos. En términos de número de pasos realizados por el agente, se puede concluir que ambos modelos obtienen soluciones similares. Pero en el caso de que las colisiones se consideren, el agente entrenado por nuestro modelo muestra claramente un desempeño mucho mejor, incluso para los peores casos representados. Para finalizar este experimento, se muestran más resultados cualitativos en el siguiente video[1].



(a) Escena del baño.

(b) Escena del dormitorio.

(c) Escena del salón.

(d) Escena de la cocina.

Figura 4.1: Número de pasos y colisiones acumulados durante la fase de entrenamiento para cada escena. Se muestra el mejor y peor caso para cada modelo.

Observando las gráficas expuestas en la figura 4.1 se confirma de nuevo la validez de la función de recompensa que se plantea para el modelo propuesto en este TFM. Si uno se fija detenidamente en las gráficas de colisiones para los mejores casos, se aprecia un

[1] https://www.youtube.com/watch?v=Eyxw-FY-iM0

espacio significante entre ambos modelos, pudiendo haber varias magnitudes de unidad de diferencia entre el modelo planteado y el modelo propuesto por Zhu *et al.* [2]. Por tanto el modelo que se propone es capaz de prevenir colisiones de manera mucho mejor que el modelo en el que se compara este proyecto.
En cuanto al número de pasos es cierto que el modelo que se propone ejecuta un menor número de pasos, aunque ambos modelos tienen un rendimiento muy similar. Esto ocurre debido al tamaño de las escenas, de forma que cuando el agente aprende las rutas óptimas, es la capacidad de evitar colisiones del agente lo que marca la diferencia entre ejecutar un número mayor o menor de pasos durante la navegación, siendo el modelo propuesto más óptimo y por tanto realizando rutas más cortar como se puede observar.

4.2.2. Experimento de rápida convergencia

Durante el entrenamiento del modelo que proponemos en este trabajo, era posible examinar cómo evolucionaba la función de recompensa en función del número de *fotogramas* procesados a lo largo del tiempo. Durante la evaluación se observó un fenómeno que resultó ser de interés para el experimento que se expone en esta sub-sección. El modelo que exponemos en este TFM es capaz de converger, al menos en una solución sub-óptima, mucho más rápido que el modelo que se propone en Zhu *et al.* [2].

Para probar lo mencionado anteriormente de una manera cuantitativa, procedimos a evaluar el desempeño de navegación de los dos modelos para cinco millones y diez millones de iteraciones de entrenamiento (en este caso, nos referimos al número de *fotogramas* que se procesan para entrenar el modelo). En la figura 4.2 se puede apreciar el número de pasos medios y colisiones realizadas por el agente. Uno puede concluir que el modelo que se propone aprende a resolver el problema planteado de manera más rápida. Es interesante observar, cómo el número de pasos indicados para $5M$ es bastante similar el indicado para $10M$ para el modelo que se presenta en este proyecto. En el caso del modelo de Zhu *et al.* [2], se aprecia que el número de pasos puede optimizarse al procesar un número mayor de *fotogramas*. Por otro lado, respecto al número de colisiones, se destaca que aunque el modelo de Zhu *et al.* [2] es capaz de minimizarlas de manera indirecta, el modelo que proponemos reduce las colisiones de manera drástica incluso para fases iniciales del proceso de aprendizaje. Como conclusión: la solución para la anticipación de colisiones que se propone no solo mejora los resultados en términos de número de pasos dados por el agente y colisiones, sino que también aprende de manera más rápida.

4.2.3. Experimento con objetivos difíciles

Para todos los experimentos de navegación mostrados en la sección anterior, se tomó estrictamente la configuración experimental original propuesto y proporcionado por Zhu *et al.* [2], usando un número fijo de cinco *objetivos* diferentes por cada una de las escenas recogidas en la base de datos propuesta por los autores. En este experimento proponemos

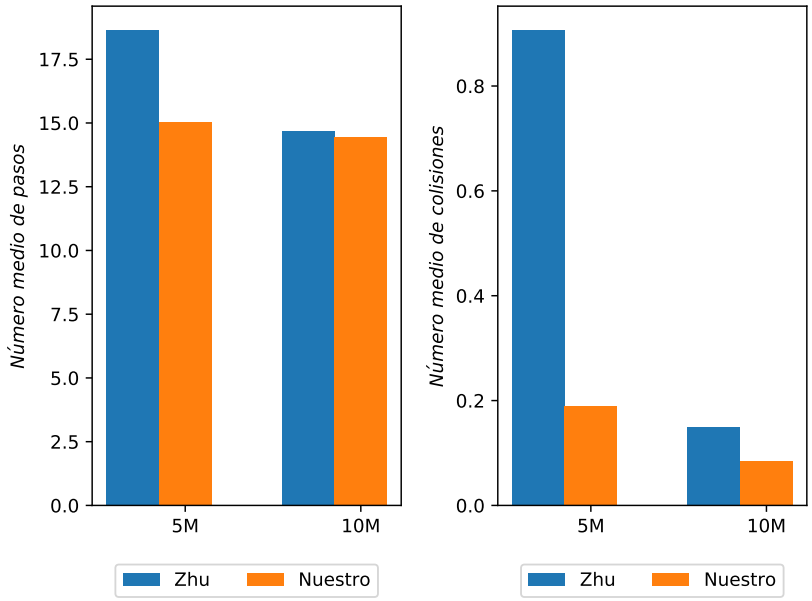


Figura 4.2: Número medio de pasos y colisiones experimentadas por el agente en el experimento de convergencia.

realizar una evaluación de ambos modelos cuando otra serie de objetivos se tienen en cuenta durante el entrenamiento. Por tanto, escogimos un nuevo conjunto de *objetivos* (figura 4.3), con el fin de dar robustez al principal objetivo que presentamos en este trabajo: anticipación de colisiones para sistemas de navegación visual. Para este propósito, en este experimento cinco nuevos *objetivos* fueron escogidos para cada escena, que en términos de colisiones y navegación, son más complicados. En la figura 4.3 se observa que los nuevos objetivos alcanzan un estado final de colisión, es decir, el agente ha de llegar hasta un objeto que le impida avanzar. Sin embargo, los objetivos empleados por el modelo base carecen de esa característica, pues como se observa en la figura 4.3, muchos de ellos implica al agente desplazarse hasta mitad de un pasillo, lo cual facilita en una mayor proporción la tarea de navegación.

Los métrica resultante para este experimento quedan reflejados en la tabla 4.2. Los resultados obtenidos para ambos modelos, en cuanto a número de pasos y colisiones se refiere, decrementa, confirmando que el nuevo conjunto de objetivos seleccionado son más difíciles de alcanzar de lo que eran los del experimento de la sección anterior.

Observando los resultados obtenidos, se recalca que durante la evaluación realizada, el agente es lanzado en posiciones aleatorias dentro de la escena, midiendo el número de pasos y colisiones sufridas durante el proceso de navegación hasta que el agente es capaz de alcanzar el objetivo marcado. En el caso de la escena etiquetada como *salón*, se destaca que es una escena que contiene diferentes objetos, como por ejemplo sofas, tablas, sillas, etc, que dificultan la navegación para ambos modelos, así como se muestra claramente en los resultados. Uno puede observar que el rendimiento del modelo que se presenta en este

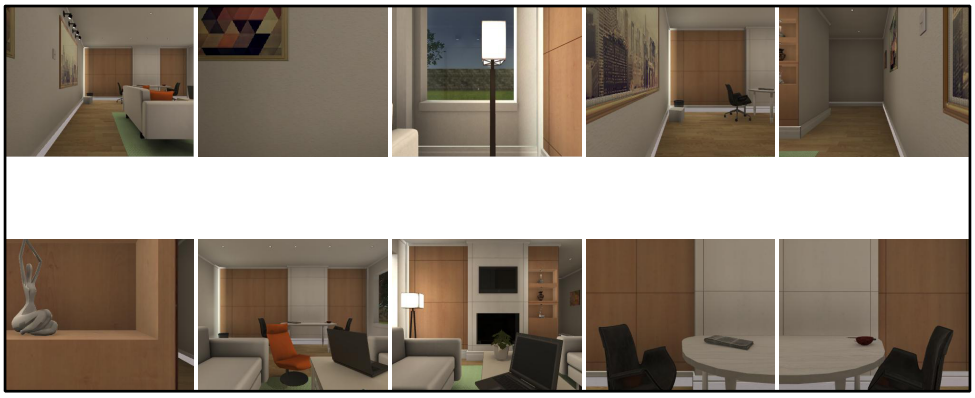


Figura 4.3: La fila superior contiene los objetivos empleados inicialmente por Zhu *et al.* [2]. La fila inferior muestra los nuevos objetivos propuestos en este proyecto. Se resalta la dificultad para alcanzar los nuevos objetivos frente a los empleados por el modelo base.

	Escena	Baño	Dormitorio	Salón	Cocina	Promedio
Zhu [2]	Pasos	7.43	13.76	1323.88	19.17	341.06
	Colisiones	0.06	0.15	247.78	0.30	62.07
Nuestro	Pasos	7.29	14.63	568.78	18.89	152.39
	Colisiones	0.03	0.08	61.66	0.25	15.5

Tabla 4.2: Evaluación del modelo para el experimento de objetivos difíciles. 10M de *fotogramas* procesados durante la fase de entrenamiento.

proyecto para la escena *salón* es mucho mejor que el reportado por el modelo de Zhu *et al.* [2]. Aún así, el número de pasos necesarios para esta escena es superior a 500 en el caso del modelo presentado en este TFM. Sin embargo, uno debe apreciar la gran diferencia en resultados entre el modelo expuesto en este trabajo y el modelo de Zhu *et al.* [2]. Las trayectorias obtenidas para nuestro modelo en promedio son más cortas y sufren menos colisiones. Si para este nuevo experimento se quiere que el rendimiento sea el mismo que el mostrado en la sección 4.2.1 (mostrados en la tabla 4.1), se deben entrenar los modelos de manera que se procesen $20M$ de *fotogramas*, así como se muestra en la tabla 4.3.

	Escena	Baño	Dormitorio	Salón	Cocina	Promedio
Zhu [2]	Pasos	7.46	13.92	18.12	18.03	14.38
	Colisiones	0.04	0.12	0.2	0.19	0.14
Nuestro	Pasos	7.16	14.15	17.73	17.86	14.225
	Colisiones	0.01	0.03	0.05	0.07	0.04

Tabla 4.3: Evaluación del modelo para el experimento de objetivos difíciles. 20M de *fotogramas* procesados durante la fase de entrenamiento.

Tras mostrar los resultados expuestos en la tabla 4.3 se concluye que no solo influye la dificultad de los objetivos a los que se quiera navegar, sino también el tiempo de procesa-

miento otorgado al agente para aprender a navegar. Es decir, el número de *fotogramas* que puede procesar el agente para alcanzar los objetivos marcados. Esta reflexión surge a raíz de comparar la mejora entre las tablas 4.2 y 4.3, donde se permite al agente procesar un mayor número de fotogramas, pero también al ver una ligera mejora entre las tablas 4.3 y 4.1, donde se han entrenado para objetivos distintos y aún así se han logrado mejores resultados siendo los objetivos más difíciles de alcanzar, siendo el caso de la tabla 4.3. Concluyendo que el número de *fotogramas* influye en alcanzar buenos resultados.

4.3. Experimentos de generalización

La capacidad de generalización de un modelo basado en *deep* RL es un aspecto interesante a evaluar. El término de generalización aplicado al problema que se intenta resolver en este proyecto hace referencia a la capacidad del agente de alcanzar *objetivos* que el modelo no haya visto durante la fase de entrenamiento. Por tanto, en esta sección, se propuso dar respuesta a la siguiente pregunta: ¿es capaz el agente entrenado de navegar hacia *objetivos* que no hayan sido considerados durante el entrenamiento? Para ello, en esta sección diseñamos una serie de experimentos para evaluar si el agente entrenado en el modelo que se presenta en este TFM dota de dicha capacidad de generalización al agente.

Para evaluar la capacidad de generalización es necesario tener una configuración experimental diferente a la que se ha tenido hasta ahora. En los experimentos anteriores, se dispone de una lista de *objetivos* que el modelo usará para entrenar y optimizar las políticas de movimiento. Éstos mismos *objetivos* serán los que son usados para realizar una evaluación y reportar los resultados mostrados en las secciones anteriores. Sin embargo, en esta sección se disponen de dos conjuntos de *objetivos*. Un conjunto se utilizará para entrenar el modelo, y el otro se empleará para evaluar el mismo. Es decir, la lista de objetivos usados en la fase de evaluación nunca son observados por el modelo durante la fase de entrenamiento.

4.3.1. Evaluación con *objetivos* cercanos a los usados durante el entrenamiento

En Zhu *et al.* [2] se juzga la capacidad de generalización, evaluando si el modelo es capaz de llegar a objetivos cercanos a los empleados para entrenar el modelo. Esto es debido a que el modelo realiza un mapeo de las imágenes de entrada en un espacio bidimensional. Por tanto, *objetivos* que se localicen cerca en el entorno, tendrá el mismo comportamiento en dicho espacio bidimensional, es decir, también estarán cercanos y por ende, el modelo es capaz de generalizar.

Tomando esta idea como base, se diseña el siguiente experimento. Se seleccionan 5 *objetivos* (diferentes a los usados para entrenar el modelo) localizados a 1, 2, 4 y 8 pasos alejados de los *objetivos* del conjunto de entrenamiento. En este experimento se hace

uso de la métrica *puntuación de éxito* mencionado en la sección 4.1. En la figura 4.4 se muestran los resultados obtenidos para este experimento. Como era de esperar, así como nos alejamos de los *objetivos* empleados para entrenar el modelo, la *puntuación de éxito* obtenido es más bajo, ya que el problema se vuelve más complejo. Sí se calcula el promedio para las cuatro situaciones (1 paso, 2 pasos, 4 pasos y 8 pasos) para sendos métodos, el modelo propuesto por [2] obtiene $pe = 57{,}4\,\%$, mientras que el modelo propuesto en este TFM reporta $pe = 58{,}1\,\%$.

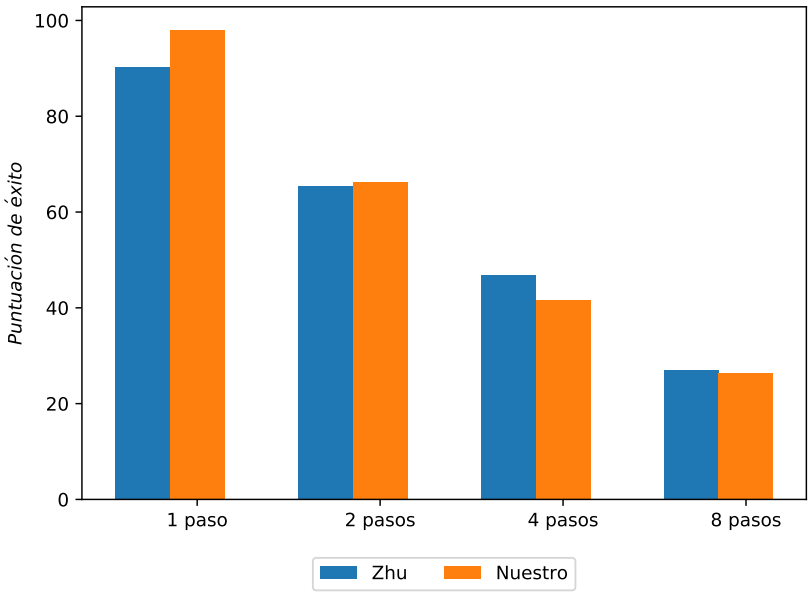


Figura 4.4: *Puntuación de éxito* para el experimento de generalización.

4.3.2. Aumento incremental del número de *objetivos* durante el entrenamiento

El segundo experimento de generalización planteado en este proyecto tiene relación con el número de *objetivos* empleados durante la fase de entrenamiento. En este experimento se quiere evaluar la repercusión que conlleva entrenar con un número determinado de *objetivos* para que un modelo basado en *deep* RL adquiera capacidades de generalización.

Para ello se diseñó el siguiente experimento. Para evaluar el modelo, se escogieron cinco *objetivos* diferentes de manera fija para cada una de las escenas de la base de datos. En cuanto a la fase de entrenamiento, se generan de manera incremental 5, 15, 25, 35 y 45 *objetivos*, con los que se entrenará el modelo. Se asegura que los *objetivos* que forman el modelo de evaluación de este experimento no se incluyen en ningún momento en los conjuntos de *objetivos* para entrenar el modelo mencionados anteriormente.

En la figura 4.5 se puede observar el valor de *sc* obtenido a medida que el número de *objetivos* para el entrenamiento se incrementa. Como es esperado, a medida que el

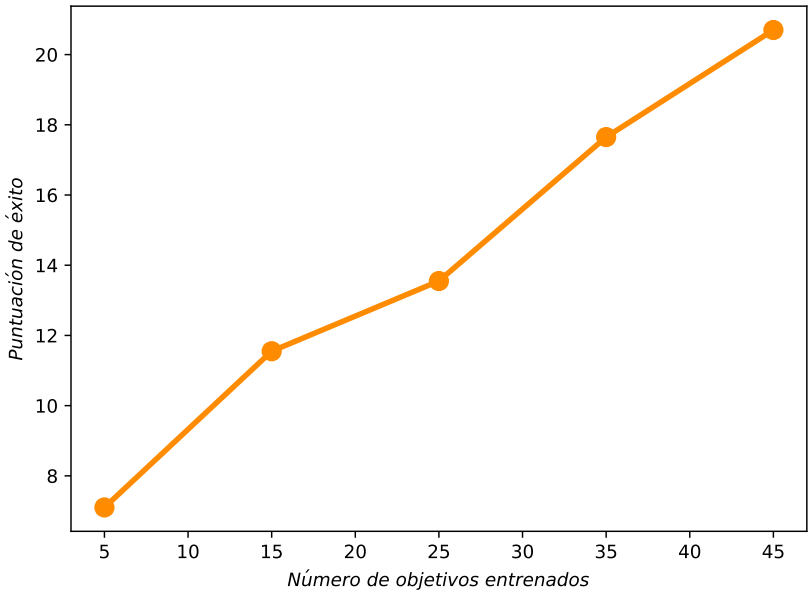


Figura 4.5: Evolución de la *puntuación de éxito* obtenida según el número de objetivos por entrenamiento es incrementado.

número de *objetivos* considerados para el entrenamiento es incrementado, el valor de *sc*, reportado por el modelo presentado en este trabajo, aumenta.

Aún así, la capacidad de generalización mostrada por el modelo que se presenta en este TFM se considera un aspecto que puede ser mejorado. Una posible solución puede consistir en realizar una revisión más exhaustiva de los *objetivos* que se generan de manera incremental, ya que al ser aleatoria, muchos de ellos puede a hacer referencia a posiciones dentro del entorno sin significado alguno para la navegación, como por ejemplo puede ser una pared.

Capítulo 5

Conclusiones y futuras líneas de trabajo

Esta sección recoge las principales conclusiones del proyecto expuesto en el presente documento. Además, se añade una sección de futuras líneas de trabajo. En ella, se exponen las posibles mejoras que puedan realizarse sobre el proyecto para un mejor funcionamiento, así como nuevas vías de investigación abiertas por el modelo presentado.

5.1. Conclusiones

Tras todo el trabajo de comprensión, desarrollo y escritura que ha supuesto este TFM, se pueden exponer las siguientes conclusiones.

En este trabajo se propone un modelo basado en *deep* RL cuyo propósito consiste en desplazar un agente o plataforma robótica hacia un objetivo definido de manera visual, de manera que el camino más corto y separado de los objetos es escogido, es decir, anticipando las posibles colisiones que se puedan originar. De acuerdo con los resultados expuestos en este documento, este TFM agrupa las siguientes contribuciones:

- Se han introducido una solución basada en *deep* RL que tiene en cuenta las colisiones durante la navegación, por medio del correcto diseño de la función de recompensa.
- Se ha realizado exhaustivos experimentos para demostrar que el modelo planteado en este proyecto mejora el estado del arte en el entorno virtual AI2-THOR en términos de número de pasos y colisiones sufridas.
- Además, los resultados muestran que el modelo presentado en este documento es capaz de aprender a resolver el problema planteado mucho antes que otro modelo que no tiene en cuenta las colisiones durante el proceso de aprendizaje.
- Finalmente, los experimentos de generalización muestran que el modelo para la anticipación de colisiones presenta una capacidad de generalización que tiene que ser mejorada para incorporar tanto nuevos objetivos como nuevas escenas, que no hayan sido considerados durante el entrenamiento, para el problema de la navegación planteado.

- Adicionalmente, el modelo de navegación capaz de anticipar colisiones, y la extensiva evaluación descrita han dado lugar a la siguiente publicación científica: [38][1].

5.2. Futuras líneas de trabajo

Esta sección alberga las posibles líneas de trabajo que deja abierto este TFM. A continuación, se enumeran tanto las posibles mejoras que se pueden implementar en el sistema presentado en este documento, como las posibles ampliaciones o aplicaciones que se pueden desarrollar a partir del modelo expuesto en este proyecto:

- Como ya se expuso en los experimentos de generalización, una de las mejoras que se puede realizar en el modelo presentado en este proyecto tiene que ver con su capacidad de generalización. Esta habilidad es muy costosa de conseguir en sistemas inspirados en la tecnología de *deep* RL. La capacidad de generalizar la navegación para nuevos objetivos o en nuevas escenas, es de vital importancia para la futura implantación de plataformas en entornos domésticos o comerciales, por ello se plantea como una futura línea de trabajo del presente proyecto.

- Este TFM ha hecho uso de las numerosas ventajas proporcionadas por el entorno virtual AI2-THOR, para resolver el problema de navegación visual. Sin embargo, la navegación por sí sola puede carecer de utilidad en algunas situaciones, por ejemplo, a la hora de ir en busca de un objeto y recogerlo. Es por ello, que otra línea de trabajo consiste en la integración de la solución para la navegación basada en la anticipación de colisiones con otras soluciones para resolver el problema de *grasping*. De manera que se puedan desarrollar plataformas completas, es decir, que no solo naveguen en busca de un objeto, sino que además sean capaces de alcanzarlo.

- La solución que se plantea en este TFM es válida en entornos virtuales. Sin embargo, transportar la solución planteada en este trabajo a entornos reales, resulta una interesante línea de investigación a seguir. Como bien se ha mencionado, el entorno virtual AI2-THOR contiene imágenes de entornos domésticos con un alto nivel de realismo. Aún así, a la hora de aplicar las políticas de movimiento aprendidas en un entorno virtual en la navegación de entornos reales, el rendimiento de los sistemas disminuye considerablemente. Esto ocurre por la discrepancia existente entre las imágenes obtenidas de los entornos virtuales, frente a las imágenes capturadas por una plataforma robótica en un entorno real. Por ello, se propone como posible futura línea de investigación, la aplicación de técnicas conocidas como adaptación de dominio, o DA, para implantar las políticas de movimiento en entornos reales.

[1] http://www.ibpria.org/2019/

Bibliografía

[1] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017. IX, XVII, 13, 14

[2] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. In *IEEE International Conference on Robotics and Automation*, 2017. IX, XVIII, XX, XXI, XXI, XXI, 1, 11, 14, 21, 22, 25, 26, 27, 28, 30, 31, 32

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013. XVII, 7

[4] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179–1187, Sep. 1989. XVII, 1

[5] Johann Borenstein and Yoram Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *Robotics and Automation, IEEE Transactions on*, 7:278 – 288, 07 1991. XVII, 1

[6] D. Wooden. A guide to vision-based map building. *IEEE Robotics Automation Magazine*, 13(2):94–98, June 2006. XVIII, 1

[7] Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera. In *ICCV*, volume 2, pages 1403–1410, 01 2003. XVIII, 1

[8] M. Tomono. 3-d object map building using dense object models with sift-based recognition features. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1885–1890, Oct 2006. XVIII, 1

[9] Kiyosumi Kidono, Jun Miura, and Yoshiaki Shirai. Autonomous visual navigation of a mobile robot using a human-guided experience. *Robotics and Autonomous Systems*, 40(2):121 – 130, 2002. Intelligent Autonomous Systems - IAS -6. XVIII, 1

[10] E. Royer, J. Bom, M. Dhome, B. Thuilot, M. Lhuillier, and F. Marmoiton. Outdoor autonomous navigation using monocular vision. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1253–1258, Aug 2005. xviii, 1

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. xx

[12] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *33rd International Conference on International Conference on Machine Learning*, ICML'16, pages 1928–1937. JMLR.org, 2016. xx, 10, 22

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. 2

[14] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, June 2006. 6

[15] Tom Duckett and Ulrich Nehmzow. Mobile robot self-localisation and measurement of performance in middle-scale environments. *Robotics and Autonomous Systems*, 24, 05 2000. 6

[16] Xiaorui Zhu, Chunxin Qiu, and Mark Minor. Terrain inclination aided three dimensional localization and mapping for an outdoor mobile robot, 05 2019. 6

[17] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992. 6

[18] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145 – 155, 1992. Range Image Understanding. 6

[19] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152, Oct 1994. 6

[20] Ashwary Anand, Shubh Agrawal, Shivang Agrawal, Aman Chandra, and Krishnakant Deshmukh. Grid-based localization stack for inspection drones towards automation of large scale warehouse systems. 06 2019. 6

[21] Saurabh Gupta, David Fouhey, Sergey Levine, and Jitendra Malik. Unifying map and landmark based representations for visual navigation. 12 2017. 6

[22] I. Ohya, A. Kosaka, and A. Kak. Vision-based navigation by a mobile robot with obstacle avoidance using single-camera vision and ultrasonic sensing. *IEEE Transactions on Robotics and Automation*, 14(6):969–978, Dec 1998. 6

[23] Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, pages 1017–1023, Cambridge, MA, USA, 1995. MIT Press. 7

[24] Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In Marcelo H. Ang and Oussama Khatib, editors, *Experimental Robotics IX*, pages 363–372, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. 7

[25] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23(2):279–303, May 1996. 7

[26] Hajime Kimura, Toru Yamashita, and Shigenobu Kobayashi. Reinforcement learning of walking behavior for a four-legged robot. *IEEE Transactions on Electronics, Information and Systems*, 122(3):330–337, 2002. 7

[27] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013. 7

[28] Lei Tai and Ming Liu. Deep-learning in mobile robotics - from perception to control systems: A survey on why and why not. *ArXiv*, abs/1612.07139, 2016. 7

[29] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L. Cun. Off-road obstacle avoidance through end-to-end learning. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 739–746. MIT Press, 2006. 7

[30] L. Tai, S. Li, and M. Liu. A deep-network solution towards model-less obstacle avoidance. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2759–2764, Oct 2016. 7

[31] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2722–2730, Dec 2015. 7

[32] Muhammad Usama and Dong Eui Chang. Robotic navigation using entropy-based exploration. *CoRR*, abs/1906.06969, 2019. 11

[33] Liulong Ma, Yanjie Liu, Jiao Chen, and Dong Jin. Learning to navigate in indoor environments: from memorizing to reasoning. *CoRR*, abs/1904.06933, 2019. 11

[34] Fengda Zhu, Linchao Zhu, and Yi Yang. Sim-real joint reinforcement transfer for 3d indoor navigation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 12

[35] Linhai Xie, Sen Wang, Andrew Markham, and Agathoniki Trigoni. Towards monocular vision based obstacle avoidance through deep reinforcement learning. *ArXiv*, abs/1706.09829, 2017. 12

[36] Jingwei Zhang, Lei Tai, Peng Yun, Yufeng Xiong, Ming Liu, Joschka Boedecker, and Wolfram Burgard. Vr-goggles for robots: Real-to-sim domain adaptation for visual control. *IEEE Robotics and Automation Letters*, 4(2):1148–1155, 2019. 12

[37] Joshua Tobin, Rachel Fong, Alex Ray, Juliane Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017. 12

[38] Eduardo Gutiérrez-Maestro, Roberto López-Sastre, and Saturnino Maldonado-Bascón. Collision anticipation via deep reinforcement learning for visual navigation. *Iberian Conference on Pattern Recognition and Image Analysis*, 07 2019. 36

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR