Universidad de la República
Facultad de Ingeniería

SLAM ACTIVO con APRENDIZAJE REFORZADO PROFUNDO

Proyecto de Grado - Estado del Arte

Autores:
Alcalde, Martín
Ferreira, Matías
González, Pablo

Tutores:
Andrade, Federico
Tejera, Gonzalo

Agosto, 2021

Índice de figuras

Siglas

ARAS Ambiguity-aware Robust Active SLAM.

DDPG Deep Deterministic Policy Gradient.

DL Deep Learning.

DQN Deep Q-Network.

DRL Deep Reinforcement Learning.

DRQN Deep Recurrent Q-Network.

LiDAR Light Detection And Ranging.

MDP Markov Decision Process.

MH-SLAM Multi Hypothesis SLAM.

RBPF Rao-Blackwellized Particle Filtering.

RL Reinforcement Learning.

RNN Recurrent Neural Network.

ROS Robot Operating System.

SLAM Simultaneous Localization and Mapping.

Índice

1. Introducción

1.1. Fundamentos teóricos

1.1.1. SLAM

Simultaneous Localization and Mapping (SLAM) [17] aborda el problema de un robot que navega en un entorno desconocido, comenzando en una ubicación con coordenadas conocidas. Su movimiento es incierto, lo que dificulta gradualmente determinar sus coordenadas globales. Mientras deambula, el robot puede percibir su entorno. El problema de SLAM es el problema de construir un mapa del entorno mientras se determina simultáneamente la posición del robot en relación con este mapa.

Formalmente, SLAM se describe mejor en terminología probabilística [17]. En ese sentido, hay dos formas principales del problema SLAM [18], que tienen la misma importancia práctica. Uno se conoce como el problema SLAM en línea que implica estimar el posterior sobre la pose momentánea junto con el mapa:

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{1}$$

aquí x_t es la pose en el tiempo t, m es el mapa y $z_{1:t}$ y $u_{1:t}$ son las medidas y los controles, respectivamente. Este problema se denomina problema SLAM en línea ya que solo involucra la estimación de variables que persisten en el tiempo t.
El segundo problema de SLAM se denomina problema de SLAM completo. En SLAM completo, buscamos calcular un posterior sobre toda la ruta $x_{1:t}$ junto con el mapa, en lugar de solo la pose actual x_t:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{2}$$

1.1.2. SLAM activo

SLAM activo [12] es un problema de toma de decisiones en el que se elige la trayectoria de un robot tanto para mejorar los resultados del mapeo y localización, como para realizar otras tareas como la cobertura o la exploración.

El robot explora activamente su entorno en busca de un mapa preciso [17]. Los métodos de SLAM activo tienden a producir mapas más precisos en menos tiempo, pero limitan el movimiento del robot.

1.1.3. RBPF

Rao-Blackwellized Particle Filtering (RBPF) es un algoritmo de SLAM activo utilizado construir mapas de entornos [19] mediante la estimación de la pose del robot independientemente de la parte posterior del mapa. Esto se muestra en la Ecuación (3):

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = p(m|x_{1:t}, z_{1:t})p(x_{1:t}|z_{1:t}, u_{t-1}) \tag{3}$$

donde $x_{1:t}$ es la trayectoria del robot, $z_{1:t}$ es la secuencia de medidas del sensor, u_{t-1} es la entrada de control en el paso de tiempo anterior y m es el mapa del entorno.

1.1.4. Entropía del Mapa

La entropía del mapa [20] corresponde a la suma de la entropía de todas las celdas c en el mapa m y cuantifica las incertidumbres en el mapa. Una buena política de exploración es una política que explora el entorno y reduce las incertidumbres. Se representa mediante la Ecuación (4):

$$H(m) = \sum_{c_f \in m} p(c)log(p(c)) + \sum_{c_o \in m} (1 - p(c_o))log(1 - p(c_o)) \tag{4}$$

donde c_f corresponde a las celdas desconocidas y desocupadas en el mapa y c_o a las ocupadas.

1.1.5. MDP

Markov Decision Process (MDP) [18] es una tupla $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ donde $\mathcal{S}$ es el conjunto de estados posibles, $\mathcal{A}$ es el conjunto de acciones, $\mathcal{P}$ describe la dinámica del entorno, es decir, la probabilidad de transición de un estado s_t al siguiente estado s_{t+1} al aplicar una acción a_t y una función de recompensa $\mathcal{R}$. Por lo general, $\mathcal{P}$ y $\mathcal{R}$ son desconocidos de antemano y deben aprenderse por ensayo y error.

1.1.6. RL

Reinforcement Learning (RL) [18] tiene como objetivo resolver un problema de toma de decisiones secuencial mediante el aprendizaje de las acciones a realizar a través de la interacción con un entorno desconocido. Al agente RL, es decir, el que toma las decisiones, no se le enseña cuál es el mejor comportamiento, sino que tiene que inferirlo recibiendo solo un valor escalar, la recompensa, por cada acción realizada. La interacción entre el agente y el entorno se puede modelar como un MDP.

1.1.7. DQN

Para elegir la mejor acción, la función de valor es estimada y empleada por muchos algoritmos RL. La función de valor contiene información sobre la calidad y conveniencia de los estados. Deep Q-Network (DQN) [15] estima la función de valor de acción de estado Q, es decir, una función que mide qué tan bueno es elegir una determinada acción en un estado dado, por medio de una red neuronal. Es común emplear aproximadores de funciones cuando el espacio de estados y/o el espacio de acción son continuos y no pueden representarse mediante una tabla de consulta. En particular, la red Q está entrenada para minimizar el error cuadrático medio entre la función de valor de acción de estado verdadero y su predicción.

1.1.8. DDPG

Deep Deterministic Policy Gradient (DDPG) [13] es un algoritmo de RL libre de modelos y fuera de la política con una estructura de actor-crítico. El actor, generalmente representado por una red neuronal, elige una acción para cada estado dado s_t. Por otro lado, el crítico, representado también por una red neuronal, evalúa las actuaciones del actor estimando la función de valor de acción del estado Q como ocurre en el algoritmo DQN [15]. DDPG puede verse como la extensión de DQN para espacios de acción continua.

1.1.9. DRQN

DQN ha demostrado tener éxito en muchas aplicaciones, sin embargo, la mayoría de ellas se basan en la suposición de Markov de que el futuro es independiente del

pasado dado el presente, es decir, la información transportada por el estado en el paso de tiempo actual es suficiente para elegir la mejor acción. En muchas situaciones, el estado del entorno no es directamente observable, es decir, en el caso de los MDP parcialmente observables (por ejemplo, dos lecturas de sensores pueden parecer iguales, pero corresponden a dos situaciones diferentes), y DQN tiene dificultades para aprender la política óptima. Para abordar este problema, se introdujo Deep Recurrent Q-Network (DRQN) [14]. DRQN extiende DQN a entornos parcialmente observables al agregar una RNN en la Q-Network.

1.1.10. DRL

Los métodos de Deep Reinforcement Learning (DRL) [10] son obtenidos al utilizar redes neuronales profundas para aproximar cualquiera de los siguientes componentes de RL: función de valor, $\hat{\mathrm{v}}(s;\theta)$ o $\hat{\mathrm{q}}(s,a;\theta)$, política $\pi(a|s;\theta)$ y modelo (función de transición de estado y función de recompensa). Aquí, los parámetros θ son los pesos en redes neuronales profundas.

1.2. Presentación del problema

Explorar y mapear de manera autónoma es uno de los desafíos abiertos de la robótica y la inteligencia artificial. Especialmente cuando se desconocen los entornos, elegir la directiva de navegación óptima no es sencillo [1]. Este problema generalmente se aborda utilizando algoritmos de planificación de rutas basándose en representaciones de los entornos: los mapas. Estos mapas generalmente se construyen utilizando algoritmos de SLAM [18]. Sin embargo, en muchas aplicaciones interesantes, un mapa completo del entorno es caro de obtener o difícil de mantener actualizado.

En los últimos años, RL [11] se ha utilizado para abordar y resolver varias tareas de robótica diferentes, como estabilización, manipulación, locomoción y navegación. En el contexto de la navegación, los planificadores de rutas basados en RL no suelen depender de ningún mapa o SLAM y, aunque tienen mucho éxito, no aprovechan la información importante almacenada en los mapas.

Dentro de las clasificaciones del problema de SLAM se encuentra SLAM activo. SLAM activo tiene como objetivo encontrar la secuencia de entradas de control, es decir, acciones, para explorar y construir completamente los mapas de los entornos. Este proceso secuencial de toma de decisiones se puede formular como un problema de RL, donde el agente de RL tiene que aprender las acciones que maximizan la recompensa acumulada.

En cuanto al problema de SLAM activo con RL, se plantea la necesidad de contar con funciones de recompensa que permitan:

- **Cerrado de Ciclos.** Refiere a la situación en la que el robot debe poder reconocer cuando transita por un lugar que ya ha sido visitado.
- **Reducir la Incertidumbre del Mapa.** Mediante el conocimiento de la diferencia de Entropía del Mapa por parte del robot entre los tiempos t y $t-1$, se pueden tomar acciones con el objetivo de reducir la incertidumbre del mismo.
- **Motivar la Exploración del Mapa.** Debe promover al robot a explorar lugares desconocidos en el mapa.

2. Estado del arte

En este capítulo se presentan los artículos más relevantes con la temática a abordar, realizando un análisis sobre sus contribuciones, resultados y conclusiones.

El proceso de selección de artículos constó de varias iteraciones, en una primera instancia exploratoria se seleccionaron artículos de SLAM activo, en una segunda iteración se optó dentro de SLAM activo por aquellos que resolvieran el problema mediante el uso de DRL. Posteriormente, en una tercera iteración se seleccionaron papers que cumplan con las condiciones de que la función de recompensa motive al robot a explorar, se base en la entropía del mapa o en cerrar ciclos. Por último, sólo se consideraron artículos reciententes (menos de tres años desde su publicación).

Cabe recalcar que los artículos presentados a continuación están en orden cronológico de publicación, siendo el último el más actual.

2.1. A survey on visual navigation for artificial agents with DRL

En [8] se realiza una revisión sobre el estado del arte de la navegación visual dado que, las que había no eran exaustivas o sistemáticas. Además es un campo que se mueve con rapidez y surgen nuevos descubrimientos.

La navegación visual (vNavigation) es una tecnología clave y fundamental para la interacción de los agentes artificiales con el entorno para lograr comportamientos avanzados. La navegación visual para agentes artificiales con DRL es un nuevo punto de acceso de investigación en inteligencia artificial y robótica que incorpora la toma de decisiones de DRL en la navegación visual. La navegación visual a través de DRL, un método de extremo a extremo, recibe directamente las imágenes de alta dimensión y genera una política de navegación óptima. En este artículo, primero presentan una descripción general sobre RL, DL y DRL. Luego, describen sistemáticamente cinco categorías principales de navegación visual DRL: vNavigation DRL directo, vNavigation DRL jerárquico, vNavigation DRL multitarea, vNavigation DRL de inferencia de memoria y vNavigation DRL de lenguaje de visión. Estos algoritmos de navegación visual DRL son revisados en detalle. Finalmente, discuten los desafíos y algunas posibles oportunidades para la navegación visual DRL para agentes artificiales.

2.1.1. Categorías en DRL vNavigation

- **Direct DRL vNavigation.** Algunos investigadores utilizan directamente algoritmos DRL en la navegación visual, donde un agente artificial se mueve en un entorno para encontrar el objeto objetivo para obtener recompensas. Durante el proceso de interacción, el agente puede aprender a navegar en estos entornos con recompensas esparzas.

- **Hierachical DRL vNavigation.** En muchos entornos dinámicos y complejos que tienen un espacio de estados de gran dimensión, los agentes artificiales a través de la navegación virtual DRL directa se enfrentarían a un desastre dimensional y serían incapaces de realizar una navegación eficaz. Para resolver el desastre dimensional, algunos investigadores propusieron una vNavigation DRL jerárquica. La navegación virtual DRL jerárquica descompone la navegación visual en subproblemas y resuelve cada uno de ellos para generar una política de navegación global basada en el principio RL jerárquico. Clasifican la vNavigation DRL jerárquica en dos tipos: vNavigation DRL de máquinas abstractas jerárquicas (HAM) y option vNavigation DRL.

- **Multi-task DRL vNavigation.** Los agentes DRL tradicionales pueden navegar bien en un dominio, pero funcionarán mal en otros dominios invisibles, lo que significa que la navegación DRL tradicional carece de transferibilidad. Para abordar el problema de la transferibilidad, se utiliza un agente DRL multitarea para adquirir parámetros de red neuronal compartidos a partir de tareas relacionadas, y los parámetros representan conocimientos de navegación compartidos. El aprendizaje de DRL multitarea puede mejorar la eficiencia de los datos y mejorar la transferibilidad de la navegación para agentes artificiales.

- **Memory-inference DRL vNavigation.** La memoria puede mejorar la capacidad de razonamiento de los agentes artificiales, lo que beneficia a la mejora del rendimiento de la navegación. Una memoria interna común es LSTM/GRU, cuya capacidad limitada restringe la capacidad de navegación del agente a entornos simples. Además, la memoria interna común combina la capacidad de cálculo y de memoria en los pesos de la red, y esta propiedad da como resultado un gran aumento en los parámetros de la red cuando aumentan las demandas

de memoria de una tarea de navegación. Para evitar las dificultades de entrenamiento de la red causadas por un gran número de parámetros, algunos investigadores han propuesto la memoria externa (replay buffer, memory networks, episodic memory, neural turing machines (NTM) / differential neural computer, etc.).

- **Vision-and-Language DRL vNavigation.** En los últimos años, un número creciente de investigadores han prestado atención a la información multimodal, que proporciona información más completa para la toma de decisiones de los agentes artificiales. Específicamente, la información multimodal más estudiada es la fusión de la visión y el lenguaje natural en el campo de la navegación.

2.1.2. Desafíos actuales de visual DRL

En general, los entornos suelen ser complejos, dinámicos y escasos de recompensas. Por lo tanto, existen dos desafíos principales a los que se enfrentan los agentes artificiales:

- **Datos ineficientes** Como las entradas del agente de navegación son imágenes de alta dimensión, el agente de navegación necesita una gran cantidad de interacciones dentro del entorno cuando aprende a navegar en un entorno.

- **Generalización pobre** La mala generalización es otro problema de DRL vNavigation. Hay dos tipos de casos de generalización: generalización de un entorno de simulación a otro entorno de simulación y generalización de un entorno de simulación a un entorno de realidad. En general, el entorno de la realidad es más complejo y dinámico, por lo que esta última generalización es más difícil.

2.1.3. Oportunidades

Cómo resolver los dos problemas anteriores es el punto clave de investigación de la navegación visual DRL. Los autores enumeran las posibles oportunidades de investigación.

- **POLICY HIERARCHY** En muchos entornos dinámicos y complejos que tienen un espacios de estado de alta dimensión, los agentes artificiales a través

de la navegación virtual DRL directa se enfrentarían a un desastre dimensional y no podrían realizar una navegación eficaz. Para resolver el desastre dimensional, los investigadores pueden utilizar la idea de jerarquía de policy. La jerarquía de policy descompone la navegación visual en subproblemas que son tareas relativamente simples para los agentes artificiales.

- **META LEARNING** En el metaaprendizaje, el objetivo del modelo entrenado es aprender rápidamente una nueva tarea a partir de una pequeña cantidad de datos.

- **MEMORY** La memoria puede mejorar la capacidad de razonamiento del modelo de navegación visual. Por lo tanto, es útil almacenar experiencias de navegación en arquitecturas de memoria y acrecentar los agentes artificiales con funciones de memoria. La memoria mejora la eficiencia de los datos y la generalización de agentes artificiales en entornos dinámicos y complejos.

- **MULTI-MODAL FUSION** La fusión multimodal, como el habla, el lenguaje natural y alguna otra información del modelo (como el radar láser y la unidad de medida inercial), puede detectar suficiente información ambiental para agentes artificiales. El desarrollo de un método de fusión multimodelo eficaz mejora la eficiencia de los datos del entorno, lo que mejora la percepción de los agentes artificiales para hacer frente a entornos dinámicos y complejos. Por lo tanto, un agente de navegación DRL visual con un sentido multimodal puede aprender una mejor policy.

2.2. Path planning for Active SLAM based on DRL under unknown environments

En [7] se hace foco en mejorar la capacidad autónoma de aprendizaje y adaptabilidad de un robot en un entorno desconocido usando SLAM activo. Se utiliza una red residual totalmente convolucional (FCRN) para reconocer los obstáculos, obtener una imagen de profundidad y construir los mapas 2D del entorno. Se elabora un algoritmo para evitar obstáculos basado en DRL y se integra con SLAM activo. La planificación para esquivar un obstáculo se lleva a cabo mediante el algoritmo Due-

ling DQN en la navegación del robot mientras que, en simultáneo, se va construyendo el mapa 2D del entorno mediante FastSLAM.

El robot navega en un entorno cerrado y construye el mapa del mismo. Si detecta un obstáculo, planifica una ruta para evitarlo utilizando aprendizaje DRL antes de alcanzar la distancia mínima con el objeto. En otro caso el robot continua navegando hasta que llega a su destino.

2.2.1. Función de recompensa

El comportamiento del robot consiste en 10 acciones básicas resultado de combinar 2 posibles velocidades lineales con 5 posibles velocidades angulares. Estas 10 acciones no solo ayudan al robot a completar la tarea de planificación de forma efectiva sino que también mejoran la eficiencia del algoritmo DRL. Cuando el robot encuentra un obstáculo, es penalizado. Cuando el robot alcanza la meta, es recompensado. Tan solo siguiendo esta estrategia el robot puede aproximarse a la meta de forma gradual y, al mismo tiempo, remover los obstáculos en el proceso de aprendizaje para finalmente completar la tarea de planificación de ruta.

La función de recompensa instantánea se define como:

$$r = v * cos(\omega) * dt \tag{5}$$

Se compone de la velocidad lineal y velocidad angular por cada paso de tiempo permitiéndole al robot ir lo más rápido posible y penalizando rotaciones en el lugar. El gráfico de recompensa total es la acumulación de recompensas instantáneas por cada paso. Si se detecta una colisión, se termina instantáneamente el evento con una penalización de -10; en caso contrario se continúa hasta que alcance la cantidad máxima de pasos (500 en este experimento). Para determinar el algoritmo más adecuado se comparó la recompensa acumulada utilizando DQN, DDQN y Dueling DQN; esto está representado en la Figura 1.

2.2.2. Evaluación y resultados

Se evalúan 3 escenarios con complejidad creciente de la siguiente forma. En los mapas, la trayectoria recorrida por el robot se representa con las curvas rojas. Los puntos negros son los obstáculos y el círculo sólido negro es el robot. En los diagramas

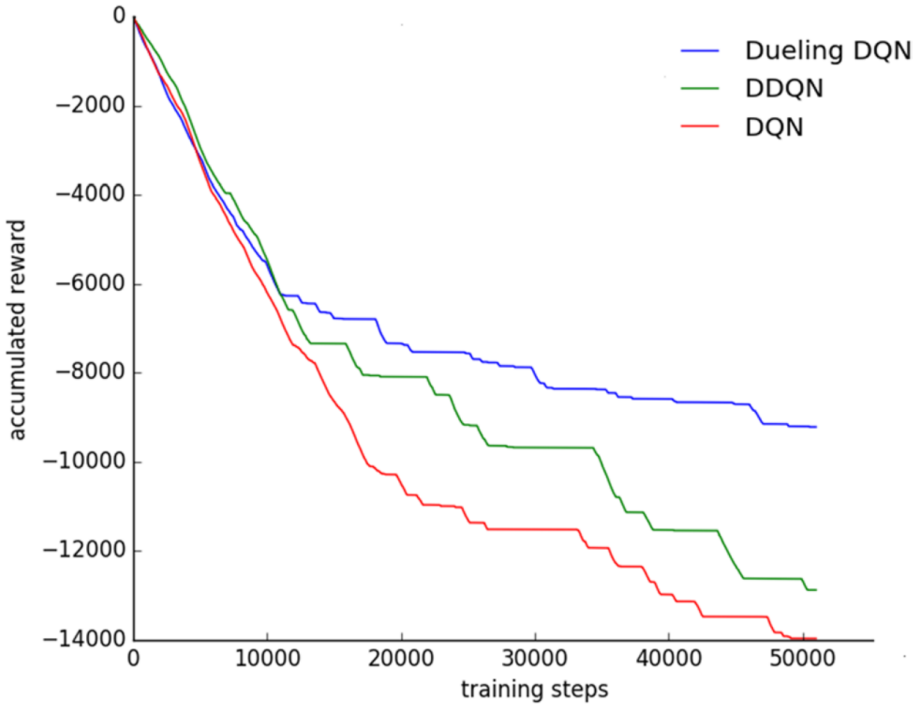


Figura 1: Curvas de aprendizaje de DQN, DDQN y Dueling DQN.

de trayectoria, las curvas rojas son las trayectorias del robot, las estrellas negras son los obstáculos y el círculo amarillo es el robot.

- Escenario 1: Dos obstáculos estáticos representado en Figura 2

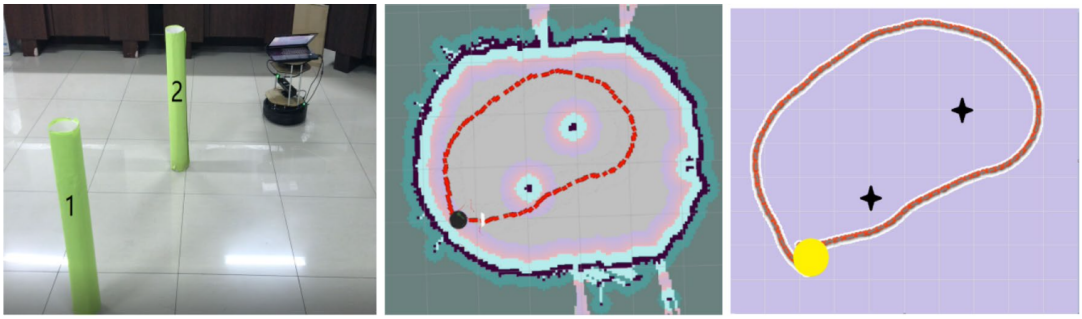


Figura 2: Corresponde a la escena **(a)**, el mapa del entorno **(b)** y la trayectoria planificada **(c)** con 2 obstáculos estáticos.

- Escenario 2: Cuatro obstáculos estáticos Figura 3

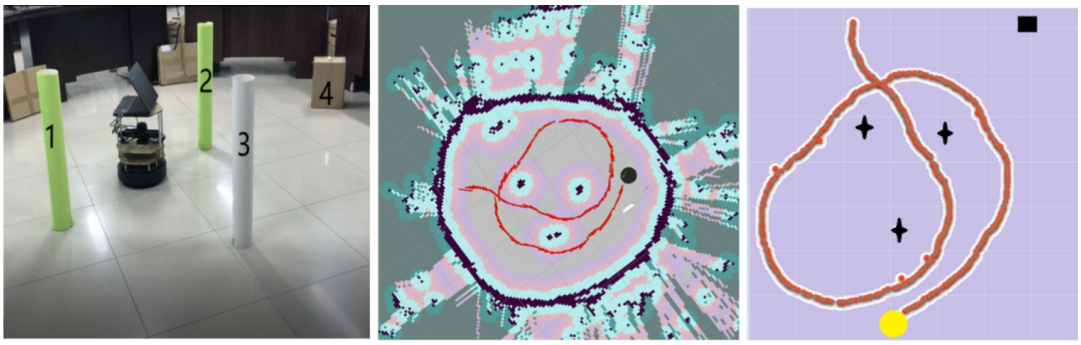


Figura 3: Corresponde a la escena **(a)**, el mapa del entorno **(b)** y la trayectoria planificada **(c)** con 4 obstáculos estáticos.

- Escenario 3: Dos obstáculos estáticos y uno dinámico Figura 4

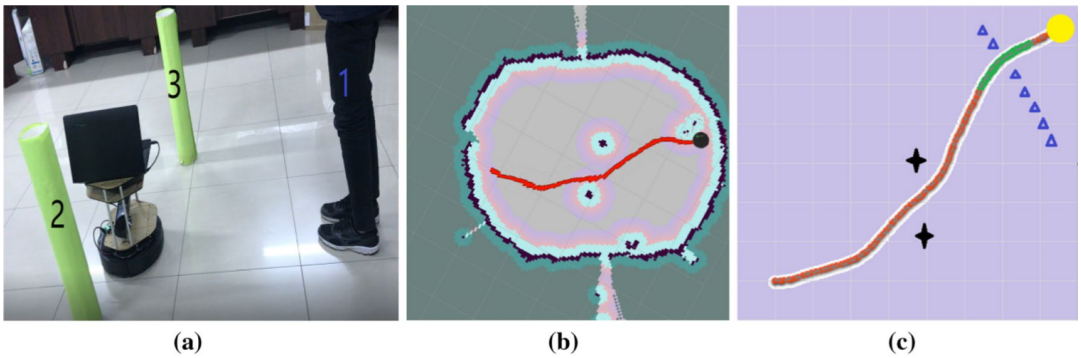


Figura 4: Corresponde a la escena **(a)**, el mapa del entorno **(b)** y la trayectoria planificada **(c)** con 2 obstáculos estáticos y 1 dinámico (una persona).

2.2.3. Conclusiones

Este paper propone path planning basado en DRL (Dueling DQN) integrado con FastSLAM sobre un entorno desconocido. DRL se usa para planificar la ruta en el entorno desconocido mientras que el algoritmo FastSLAM se usa para localizar y mapear el entorno. Tanto la simulación como los resultados experimentales muestran que este método consigue una navegación autónoma evitando obstáculos en distintos ambientes. En trabajos futuros plantean mejoras incluyendo nuevos sensores y mejorando la estrategia de aprendizaje para reducir el tiempo que lleva completar la tarea.

2.3. A DRL approach for Active SLAM

En [6] se formula el paradigma SLAM activo en términos de DRL sin modelos, incorporando las funciones de utilidad tradicionales basadas en la teoría del diseño experimental óptimo en las recompensas y, por lo tanto, relajando los cálculos intensivos de los enfoques clásicos. Validan dicha formulación en un entorno de simulación complejo, utilizando una arquitectura de Q-learning profunda de última generación con mediciones láser como entradas de red. Los agentes capacitados se vuelven capaces no solo de aprender una policy para navegar y explorar en ausencia de un modelo de entorno, sino también de transferir sus conocimientos a mapas nunca antes vistos, que es un requisito clave en la exploración robótica.

El paradigma se formula como un problema de DRL de recompensa múltiple, sobre la base de enfoques clásicos de optimización multiobjetivo y TOED. Además del simulador de Gazebo, un DQN de última generación está integrado dentro de un marco de ROS. Por tanto, se tiene en cuenta de forma conjunta el uso de una recompensa basada en la incertidumbre y un simulador complejo.

	Task Accomplished	Architecture	Reward Design	Complex Environment	Generalization Unknown env.	Partial Observability
Tai and Liu [19]	Navigation	DQN	Extrinsic	✓		
Mirowski et al. [20]	Navigation	A3C	Extrinsic	✓		
Wen et al. [22]	Navigation	D3QN	Extrinsic	✓		
Zhelo et al. [26]	Goal navigation	A3C	Intrinsic		✓	
Oh and Cavallaro [27]	Goal navigation	A3C	Intrinsic	✓		
Tai et al. [35]	Goal navigation	ADDPG	Intrinsic	✓	✓	
Zhu et al. [33]	Frontier selection	A3C	Intrinsic		✓	
Niroui et al. [34]	Frontier selection	A3C	Intrinsic	✓	✓	✓
Gottipati et al. [31]	Active localization	A2C	Posterior belief	✓	✓	
Chaplot et al. [30]	Active localization	A3C	Posterior belief	✓	✓	
Chen et al. [29]	Exploration	DQN	Entropy		✓	✓
Chen et al. [32]	Active SLAM	DQN+GNN	T-opt		✓	✓
Ours	Active SLAM	D3QN	D-opt	✓	✓	✓

Figura 5: Tabla comparativa con métodos anteriores.

2.3.1. Función de recompensa

Además, el SLAM activo se puede formular como un problema de optimización multiobjetivo [16] donde la función de costo $\mathcal{J}$ debe contener el costo de realizar una trayectoria de colisión libre ($\mathcal{F}$) y una métrica de la matriz de covarianza matrix ($\mathcal{U}$), que representa la incertidumbres en las estimaciones de la pose del robot y el mapa. Como se muestra a continuación, ambos términos deben escalarse mediante

coeficientes dependientes de la tarea y pueden considerarse en un horizonte de n pasos:

$$\mathcal{J} = \sum_n \beta_n \mathcal{F}_n + \sum_n \alpha_n \mathcal{U}_n \tag{6}$$

Dicha formulación se puede generalizar directamente al problema de RL, simplemente diseñando la función de recompensa de la siguiente manera:

$$\mathcal{R}_{\text{aug}} = \mathcal{R}_{\mathcal{F}} + \mathcal{R}_{\mathcal{U}} \tag{7}$$

Donde el primer término se refiere a una recompensa completamente extrínseca que da cuenta de las trayectorias libres de colisiones (es decir, navegación), y el segundo es una recompensa intrínseca, que se puede definir inversamente proporcional a una métrica de la matriz de covarianza:

$$\mathcal{R}_{\mathcal{F}} = \begin{cases} - \quad 100 & \text{if collision} \\ \quad 1 & \text{if } \omega = 0 \\ - \quad 0{,}05 & \text{if } \omega \neq 0 \end{cases} \tag{8}$$

$$\mathcal{R}_{aug} = \begin{cases} -100 \text{ if collision} \\ \quad 1 + \tanh\left(\dfrac{\eta}{f(\Sigma)}\right) & \text{if } \omega = 0 \\ \quad -0{,}05 + \tanh\left(\dfrac{\eta}{f(\Sigma)}\right) & \text{if } \omega \neq 0 \end{cases} \tag{9}$$

2.4. ARAS: Ambiguity-aware Robust Active SLAM based on Multi-hypothesis State and Map Estimations

En [5] presentan un framework de SLAM activo robusto consciente de la ambigüedad (ARAS) que hace uso de estimaciones de mapas y estados de hipótesis múltiples para lograr una mayor robustez.

Las mediciones ambiguas pueden resultar en múltiples soluciones probables en un sistema de Multi Hypothesis SLAM (MH-SLAM) si no se pueden resolver temporalmente (debido a información insuficiente). ARAS apunta a tener en cuenta todas estas estimaciones probables explícitamente para la toma de decisiones y la planificación. En ese sentido ARAS 1) adopta contornos locales para una exploración

eficiente de múltiples hipótesis, 2) incorpora un módulo de cierre de bucle activo que revisita las áreas mapeadas para adquirir información para la poda de hipótesis para mantener la eficiencia computacional general, y 3) demuestra cómo usar la pose de destino de salida para la planificación de trayectorias bajo las estimaciones de hipótesis múltiples. A través de extensas simulaciones y experimentos en el mundo real, el algoritmo ARAS propuesto puede mapear activamente los entornos interiores generales de manera más robusta que un enfoque similar de hipótesis única en presencia de ambigüedades.

En general, SLAM activo toma las medidas del sensor como entradas al igual que un sistema SLAM pasivo, pero genera una secuencia de decisiones / acciones en línea para influir en las medidas futuras además de las estimaciones de estado y mapa de SLAM pasivo únicamente. Las decisiones / acciones tienden a caer en uno de dos grupos complementarios: exploración y cierre de bucle activo. El objetivo de la exploración es mover el robot en el entorno para que se pueda observar nueva información. Por otro lado, el cierre de bucle activo es responsable de encontrar lugares mapeados para volver a visitar los cierres de bucle que corrigen la deriva y la incertidumbre acumuladas. La ambigüedad es la situación en la que es plausible más de una interpretación para las mismas observaciones, las cuales provienen de varias fuentes que incluyen información insuficiente, conflictos entre diferentes mediciones de sensores, asociación de datos inciertos, cierre de bucle basado únicamente en la apariencia, etc. Cuando ocurren ambigüedades, la estimación única puede estar contaminada por información incorrecta, lo que da como resultado acciones mal informadas o incluso exploración de fallas del sistema completo y cierre de bucle activo.

2.4.1. Contribución

Las contribuciones de este trabajo son:

- Introducir un framework de SLAM activo robusto consciente de la ambigüedad (ARAS, Figura 6) que utiliza el estado de hipótesis múltiples y estimaciones de mapas de MH-SLAM.
- Desarrollar un algoritmo de exploración basado en submapas libres de ambigüedad para hacer uso de las estimaciones de múltiples hipótesis de manera

eficiente.

- Diseñar la estrategia de cierre de bucle activo para reducir el número de hipótesis de tractabilidad.
- Demostrar el uso de la decisión / acción de salida para la planificación del camino considerando múltiples hipótesis.
- Evaluar el algoritmo SLAM activo a través de extensas simulaciones y discutir sus propiedades.
- Aplicar el algoritmo propuesto en el mundo real.

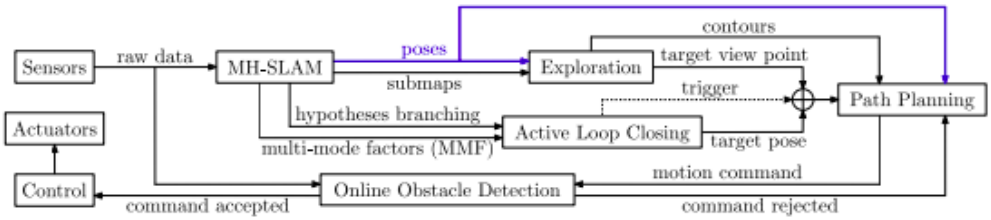


Figura 6: Diagrama del framework ARAS.

2.4.2. Propuesta

ARAS toma todas las estimaciones de hipótesis múltiples de un sistema MH-SLAM como entradas y salidas de decisiones / acciones únicas a tomar en un momento dado. Cada acción seleccionada por ARAS se incluye en una de dos categorías: exploración y cierre de bucle activo. Dado que mantener todos los mapas globales en cada hipótesis para la exploración es muy ineficiente tanto en la memoria como en el cálculo, se desarrolla un algoritmo de exploración de múltiples hipótesis basado en submapas libres de ambigüedad para lograr una mejor eficiencia. En cuanto al cierre de bucle activo, debido a que el crecimiento exponencial del número de hipótesis puede ser difícil de rastrear, el objetivo es limitar el número de hipótesis en función de las restricciones del sistema mientras se hace un seguimiento de la hipótesis correcta. Por lo tanto, el objetivo del algoritmo de cierre de bucle activo es encontrar algunos lugares para volver a visitar, de modo que los cierres de bucle detectados puedan

proporcionar información suficiente para distinguir y eliminar suficientes hipótesis erróneas para reducir el costo computacional, o incluso preservar solo la hipótesis correcta.

ARAS utiliza MHiSAM2 [9] como su solucionador de back-end, que es un optimizador no lineal incremental que calcula soluciones de hipótesis múltiples basadas en la entrada. MHiSAM2 realiza la poda automáticamente siempre que el número de hipótesis excede un umbral, la clave para evitar podar la hipótesis correcta accidentalmente es adquirir información suficiente para una poda correcta en los tiempos adecuados. Se introduce un algoritmo simple de cierre de bucle activo para cubrirlo. Suponen que n_{limit} es el límite superior del número de hipótesis que todo el sistema puede manejar y n es el número de hipótesis igualmente probables en cada iteración. Siempre que $n \geq n_{limit}$, la hipótesis correcta podría podarse accidentalmente debido a la falta de información. Por lo tanto, el cierre de bucle activo debe activarse en los momentos correctos para que los cierres de bucle se puedan detectar y registrar para proporcionar información suficiente para una poda correcta (seguir rastreando la hipótesis correcta) mientras siempre se satisface $n \leq n_{limit}$. Dado que pueden ocurrir nuevas ambigüedades en el camino para volver a visitar $M*$ y aumentar aún más n antes de que se detecte cualquier bucle positivo verdadero, el enfoque ideal es predecir ambigüedades futuras y cierres de bucles, y activar el cierre del bucle activo de antemano.

Como resultado, simplemente eligen un umbral $n_{trigger} < n_{limit}$ y activan el cierre de bucle activo cuando $n > n_{trigger}$ como enfoque de línea de base. Y si n realmente va más allá de n_{limit}, eligen podar algunas de las hipótesis antes de obtener suficiente información.

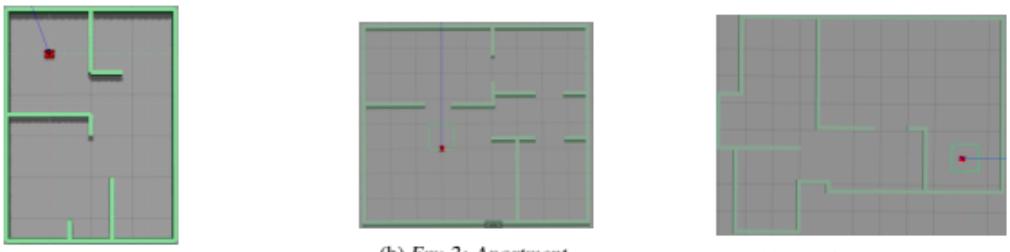
Cuando se activa el cierre del bucle activo, se seleccionará un submapa objetivo $M*$ de los submapas existentes para volver a visitarlo. Entonces, se espera que al menos un bucle que pueda resultar en una poda correcta sea detectado y cerrado cuando la pose de $M*$ se revise con éxito, o incluso antes en el camino a $M*$.

2.5. RL helps SLAM: learning to build maps

En [4] investigan el uso de RL para explorar entornos desconocidos e interiores y construir sus mapas. Para esto, se benefician del filtro de partículas RBPF como

algoritmo de SLAM para la localización y el mapeo de robots en tiempo real.

Comparan y prueban tres funciones de recompensa diferentes en diferentes entornos (Figura 7) con una complejidad creciente. Además, las funciones de recompensa son comparadas con el algoritmo de exploración basado en fronteras [21].



(a) *Env-1: Maze.* (b) *Env-2: Apartment.* (c) *Env-3: Apartment 2.*

Figura 7: Los ambientes son de $22m^2$, $68m^2$ y $65,5m^2$, respectivamente.

El desempeño de los tres planificadores de ruta diferentes basados en RL se evalúa no solo en los entornos de capacitación (Env-1 y Env-2), sino también en un entorno invisible (Env-3), a priori, para probar las propiedades de generalización de las políticas.

2.5.1. Propuesta

Consideran que la generación de señales de control y la transición de estado de SLAM resuelven un problema de RL, donde el agente tiene que aprender la mejor secuencia de acciones (la mejor secuencia de señales de control), es decir, la secuencia que maximiza la recompensa acumulada total. En este contexto, el objetivo es explorar y completar el mapa sin colisiones en el mínimo tiempo. Para elegir la mejor acción u, el agente recibe información sobre el estado del entorno y la recompensa r asociada al mismo. Debido al alto número de grados de libertad, es difícil elegir un vector de estado adecuado, pero un vector de estado informativo es crucial para un buen rendimiento del algoritmo de aprendizaje. El mismo razonamiento se aplica a la función de recompensa. La elección adecuada de la función de recompensa y el vector de estado es el enfoque de este trabajo.

2.5.1.1 Funciones de recompensa

- **Esparza.** Sencilla para aprender a mapear un entorno desconocido. Al aprender a evitar una colisión, el agente puede completar ocasionalmente el mapa.

$$R(s_t) = \begin{cases} r_{mapCompleted}, & c_t > \bar{c} \\ r_{crashed}, & s_{coll} \\ 0, & othwerise. \end{cases} \tag{10}$$

- **Completitud del Mapa.** Agrega a la función de recompensa Esparza (10) un término proporcional a la diferencia en la completitud del mapa del paso actual y el paso anterior.

$$R(s_t) = \begin{cases} r_{mapCompleted}, & c_t > \bar{c} \\ r_{crashed}, & s_{coll} \\ c_t - c_{t-1}, & othwerise. \end{cases} \tag{11}$$

  donde c_t es la completitud del mapa en el momento t y c_{t-1} es la completitud del mapa en el momento $t-1$.

- **Ganancia de Información.** Agrega a la función de recompensa Esparza (10) la diferencia en la entropía en el paso actual y en el anterior, es decir, la ganancia de información. Para esto, utiliza la ecuación de la Entropía del Mapa (4).

$$R(s_t) = \begin{cases} r_{mapCompleted}, & c_t > \bar{c} \\ r_{crashed}, & s_{coll} \\ H_t - H_{t-1}, & othwerise. \end{cases} \tag{12}$$

  donde H_t es la entropía del mapa en el momento t y H_{t-1} es la entropía del mapa en el momento $t-1$.

2.5.1.2 Vector de Estado

- Contiene solo la información actual en el momento t:

$$s_t = [z_t, a_{t-1}, x_t, c_t, t_t] \tag{13}$$

  donde s_t corresponde al vector de estado en el momento t, z_t a las lecturas actuales del sensor, a_{t-1} a la acción elegida previamente, x_t a la estimación de

la posición del robot, c_t al porcentaje de finalización del mapa y t_t a los pasos de tiempo que quedan antes del final del episodio.

- Contiene también la información del pasado reciente:

$$s_t^e = [s_{t-w}, ..., s_t] \tag{14}$$

donde s_t^e es el vector extendido que concatena la secuencia de estados pasados s, w es la longitud de la ventana de observación (cuánto del pasado se tiene en cuenta).

2.5.2. Conclusiones

En los entornos de entrenamiento, los agentes RL que explotan la información almacenada en el mapa, es decir, Completitud del Mapa (11) y Ganancia de Información (12), superan la función de recompensa Esparza (10) y el enfoque de exploración basado en fronteras en términos de longitud de ruta y, en consecuencia, tiempo para completar el mapa. La función de recompensa por Completitud del Mapa (11) logra la curva de aprendizaje más rápida sobre el entrenamiento, mientras que la recompensa basada en la Ganancia de Información (12) permite al agente aprender el camino exploratorio más seguro.

En un entorno invisible a priori con un área total similar, pero una topología diferente, los planificadores de RL entrenados con Completitud del Mapa (11) y Ganancia de Información (12) pueden completar el mapa. Sin embargo, ambos planificadores siguen caminos más largos que la exploración basada en fronteras. Los resultados dependen de la posición inicial del robot. Ganancia de Información (12) se desempeña mejor cuando se inicia en un espacio agrupado, ya que tiende a navegar hacia áreas abiertas. Completitud del Mapa (11) exhibe un comportamiento opuesto, ya que tiende a permanecer cerca de las paredes y los obstáculos.

2.6. On reward shaping for mobile robot navigation: a RL and SLAM based approach

En [3] se presenta un algoritmo de planificación de rutas sin mapas basado en DRL para robots móviles que navegan en un entorno desconocido que solo se basa en datos

de láser sin procesar de 40 dimensiones e información de odometría. El planificador es entrenado usando una función de recompensa basada en el conocimiento en línea del mapa del entorno de entrenamiento, obtenido usando el filtro RBPF basado en cuadrícula, en un intento de mejorar la conciencia de los obstáculos del agente. El agente es entrenado en un entorno simulado complejo y evaluado en dos no vistos. Se muestra que la policy entrenada utilizando la función de recompensa introducida no solo supera a las funciones de recompensa estándar en términos de velocidad de convergencia, sino que también mejora drásticamente el comportamiento de la agente en entornos no vistos. Además, la policy formada en el entorno de simulación puede transferirse directa y correctamente al robot real.

La principal novedad del artículo radica en dar forma a la función de recompensa basada en el conocimiento adquirido en línea sobre el entorno que el robot adquiere durante el entrenamiento. Este conocimiento se obtiene a través de un mapeo basado en cuadrículas con un enfoque de filtro RBPF.

De esta manera, el robot puede aprender una policy (sub) óptima en un número menor de pasos de iteración al aumentar su conocimiento sobre la ubicación de los obstáculos circundantes. Además, el agente entrenado con esta función de recompensa dependiente del mapa muestra mejores propiedades de generalización en el caso de entornos complejos e invisibles que un planificador de rutas estándar sin mapas de RL. Vale la pena mencionar que no se utiliza ningún mapa durante la evaluación del planificador de rutas RL en entornos desconocidos.

2.6.1. Función de recompensa

La meta del agente es alcanzar un objetivo deseado en el menor número de pasos de iteración evitando chocar con obstáculos. Así, el agente recibe una penalización proporcional a la exponencial negativa de la distancia euclidiana entre su posición actual y la posición de la meta; la distancia se calcula como $d = p_t^{x,y} - g_2$ donde p_t representa la posición actual del robot en el momento t con respecto al marco de inercia, y g es la ubicación del objetivo en el marco de coordenadas del robot. Además, se agrega una recompensa esparza si el agente alcanza el objetivo dentro de una tolerancia predefinida y, de la misma manera, una penalización si el robot choca con un obstáculo o excede el número máximo de pasos de iteración T permitidos en un solo episodio sin alcanzar el límite. objetivo o golpear un obstáculo. Con base en

las consideraciones anteriores, la función de recompensa general $r(s_t, a_t)$ se muestra en (15).

$$r(s_t) = \begin{cases} r_{\text{reached}}, & d \leq d_{\text{mín}}, \\ r_{\text{crashed}}, & s_{ts}, \\ 1 - e^{\gamma d}, & \text{otherwise} \end{cases} \tag{15}$$

donde γ representa la tasa de caída de la exponencial.

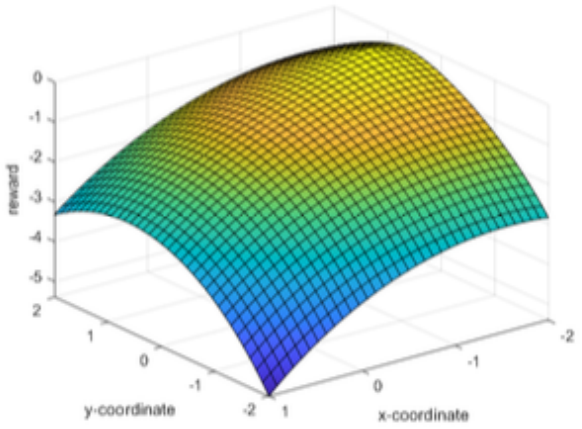


Figura 8: Una representación gráfica de la función de recompensa densa dada en (15). La penalización esparza no se muestra en aras de la claridad.

Un inconveniente de la función de recompensa densa definida en (15) es que sólo tiene en cuenta la ubicación del objetivo. En otras palabras, incluso si cambia la configuración de los obstáculos, la variedad de la función de recompensa permanecerá exactamente igual. Esto significa que el robot no aprende a ser consciente de los obstáculos dentro del entorno, sino solo a su ubicación al chocar contra ellos, debido a la penalización esparza. Por lo tanto, para mejorar el rendimiento del aprendizaje y la percepción de los obstáculos, la función de recompensa se configura en función del mapa de cuadrícula de ocupación 2D adquirido que el robot construye durante el entrenamiento. Este mapa se construye dividiendo el espacio de trabajo en el que opera el robot en celdas espaciadas uniformemente. Cada celda dentro de la cuadrícula de ocupación se clasifica como ocupada o libre según un valor de umbral predefinido que determina la probabilidad de ocupación de cada celda. Esta probabilidad se actualiza mientras el robot explora el entorno. La incorporación

del conocimiento del entorno está ponderada por el nivel de certeza de la posterior del mapa: $p\left(m \mid z_{1:t}, x_{1:t}\right) = \prod_{i=0}^{M} p\left(m_i \mid z_{1:t}, x_{1:t}\right)$. Además, dado que cada obstáculo dentro del entorno está representado por un cierto número de celdas de cuadrícula ocupadas, este término de recompensa se normaliza por el número total de celdas de cuadrícula ocupadas en el campo de visión (FOV) del robot. Intuitivamente, las áreas con mayor concentración de obstáculos devolverán más recompensas negativas que el espacio libre asumiendo la misma confianza en el mapa dada por la parte posterior del mapa. Esto se puede formular de la siguiente manera:

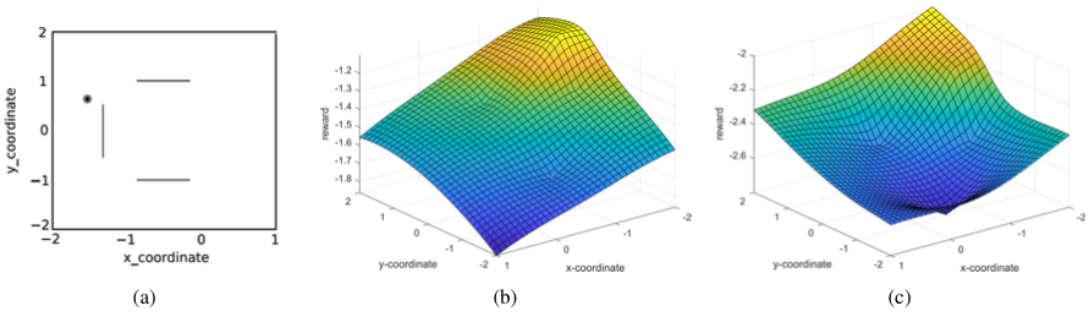
$$r_m\left(s_t, map\right) = \frac{1}{k} \prod_{i=0}^{M} p\left(m_i \mid z_{1:t}, x_{1:t}\right) \sum_{j=0}^{k} e^{-c_{j_{min}}} \tag{16}$$

donde $c_{j_{min}}$ representa la distancia entre el robot y una celda ocupada en su campo de visión. Ahora bien, la función de recompensa no depende puramente del estado, sino del mapa (distancia a las celdas ocupadas), del tiempo y de la secuencia de mediciones láser $z_{1:t}$ y poses $x_{1:t}$. Por tanto, (17) también contiene información histórica. Sin embargo, cabe mencionar que no se incluye ningún mapa o historial de medidas o poses en el vector de estado al violar parcialmente el modelo MDP, ya que la recompensa es función del estado. La elección se hace porque el objetivo final es desarrollar un planificador de rutas sin mapas. En nuestro enfoque, se construye un solo mapa y se usa sólo durante el entrenamiento del agente. Finalmente, el término dependiente del mapa definido en (16) se resta de la recompensa densa dada en (15) donde las recompensas escasas se mantienen como están. La función de recompensa propuesta se presenta en (17).

$$r\left(s_t, map\right) = \begin{cases} r_{\text{reached}}\,, & d \leq d_{\text{mín}} \\ r_{\text{crashed}}\,, & s_{ts} \\ 1 - e^{\gamma d} - r_m\left(s_t, map\right), & \text{otherwise.} \end{cases} \tag{17}$$

De esta forma, la función de recompensa no sólo depende de qué tan lejos esté el agente del objetivo, sino de la distancia a los obstáculos dentro del espacio de trabajo. Para visualizar cómo la función de recompensa varía con el cambio de la parte posterior del mapa $p\left(m \mid z_{1:t}, x_{1:t}\right)$ que aumenta gradualmente a medida que el robot se vuelve más seguro sobre el mapa, la función de recompensa (17) incluyendo el término mapa-dependiente se dibuja en dos instantes diferentes como se muestra

en la Figura 2.6.1b y 2.6.1c. Es posible notar tres áreas de depresión principales correspondientes a los tres obstáculos presentes en el entorno simple que se muestra en la Figura 2.6.1a.



(a) (b) (c)

Figura 9: La meta está ubicada en (-1.6,0.65) representada por el círculo negro en (a). La evolución del efecto del término dependiente del mapa sobre la función de recompensa, cuando $p\left(m \mid z_{1:t}, x_{1:t}\right) = 0{,}5$ en (b) y $p\left(m \mid z_{1:t}, x_{1:t}\right) = 1{,}0$ en (c). Es posible notar que las depresiones de la función de recompensa, por lo que la penalización recibida por los agentes se vuelve más severa cuanto más crece la posterior del mapa.

2.7. Entropy-based exploration for mobile robot navigation: a learning-based approach

En [2] se presenta una estrategia de exploración de un planificador DRL para aprender comandos de velocidad continua a partir de datos obtenidos de sensores y resolver tareas de navegación en espacios cerrados (desconocidos). Aborda un enfoque en el que la entropía del mapa (construido de forma online utilizando RBPF SLAM) se utiliza durante la fase de entrenamiento para modelar la función de recompensa. Las políticas aprendidas con este enfoque no solo permiten hacer una buena generalización a entornos y destinos desconocidos, sino que también pueden transferirse directamente, sin ningún ajuste adicional, a un robot real. Además, el espacio de acciones es continuo para obtener maniobras más avanzadas y mayor suavidad en las trayectorias.

En este trabajo se refuerza la conexión entre RL y SLAM sacando provecho del conocimiento almacenado en el mapa. En particular, se usa la entropía del mapa para mejorar las habilidades de exploración del planificador y su capacidad para escapar a mínimos locales que ocurren cuando los entornos son más complejos y realistas.

2.7.1. Función de recompensa

El objetivo del agente es generar comandos de velocidad para poder controlar el robot. El robot tiene que ser capaz de llegar a su destino evitando colisionar con obstáculos y quedar atrapado en mínimos locales. Para conseguir estos 3 objetivos, la función de recompensa propuesta es igual a la suma (con pesos) de 3 términos diferentes:

- **Enfocado en el destino (TD)**: El agente es recompensado si está más cerca del destino que en el paso anterior.

$$r_1(s_t) = \|p_{t-1}^{x,y} - g\|_2 - \|p_t^{x,y} - g\|_2 \tag{18}$$

- **Enfocado en evitar obstáculos (OA):** Penalización inversamente proporcional a la distancia a los obstáculos al cuadrado. Cabe mencionar que la distancia a los obstáculos es calculada en tiempo real utilizando la información del láser, no se asume como conocida de antemano.

$$r_2(s_t) = \frac{1}{(\|p_t^{x,y} - o\|_2)^2} \tag{19}$$

- **Enfocado en la entropía del mapa**: El agente es recompensado si explora el entorno o si navega por espacios abiertos y/o inexplorados anteriormente.

$$r_3(s_t) = \sum_{c_f \in m} p(c) log(p(c)) + \sum_{c_o \in m} (1 - p(c)) log(1 - p(c)) \tag{20}$$

donde c_f corresponde a las celdas desconocidas y desocupadas en el mapa y c_o a las ocupadas. Además, se agrega una recompensa dispersa $r_{reached}$ si el agente alcanza el objetivo dentro de un umbral de distancia predefinido y una penalización $r_{crashed}$ si el robot colisiona con un obstáculo o si excede la cantidad máxima de pasos T en

un episodio. De esta manera, la función de recompensa se define como:

$$R\left(s_t, m\right) = \begin{cases} r_{\text{reached}}\ , & d \leq d_{\text{mín}}, \\ r_{\text{crashed}}\ , & s_{ts}, \\ \lambda^g r_1(s_t) - \lambda^o r_2(s_t) - \lambda^H r_3(s_t), & \text{otherwise} \end{cases} \tag{21}$$

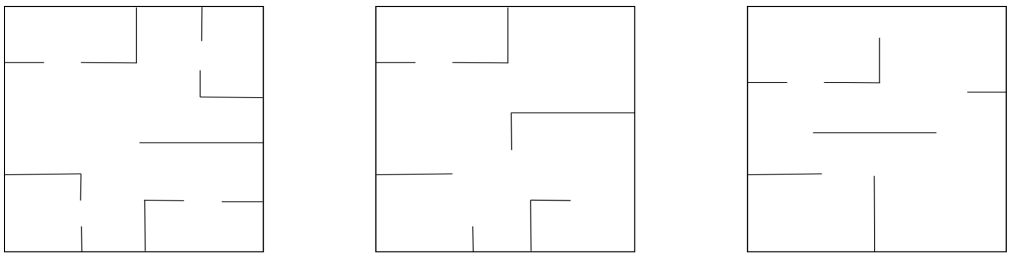
donde λ^g, λ^o, λ^H son escalares para darle peso a los términos r_i.

2.7.2. Evaluación y resultados

Utilizando la misma arquitectura de red y configuración de parámetros, se comparan los resultados entre 3 agentes con las siguientes características:

- TD: entrenado con la función de recompensa 18
- TD+OA: entrenado con la función de recompensa 18 + 19
- Propuesto: entrenado con la función de recompensa 21

A su vez, se definen 3 entornos de prueba también. Env_1 es donde se entrena a cada agente y luego la performance de las políticas son evaluadas en los entornos Env_2 y Env_3 (desconocidos de antemano).

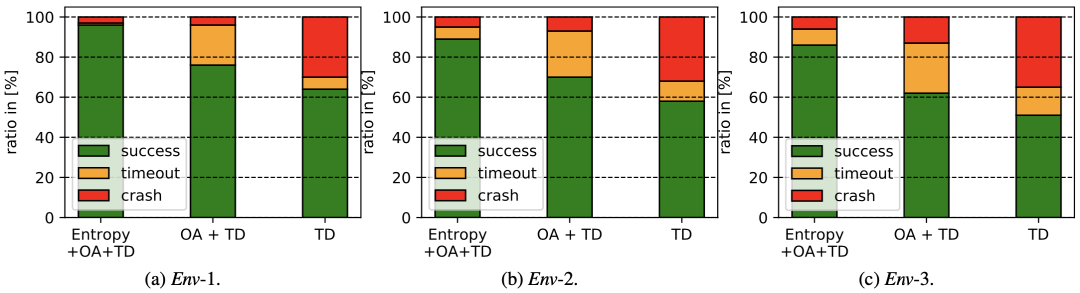


(a) *Env*-1. (b) *Env*-2. (c) *Env*-3.

Figura 10: Ambientes de entrenamiento y evaluación.

Después de haber entrenado los 3 agentes en el Env_1, las políticas aprendidas se evalúan en el mismo set de 100 objetivos generados al azar. La siguiente figura muestra la comparación de porcentaje de éxito, colisiones, timeouts (episodios finalizados sin alcanzar el objetivo ni colisionar) y el cantidad de episodios que tomaron:

El planificador entrenado con la recompensa TD pudo alcanzar el objetivo el 64 % de las veces y, en la mayoría de los episodios fallidos, colisionó con un obstáculo. El



(a) *Env*-1. (b) *Env*-2. (c) *Env*-3.

Figura 11: Comparación de evaluación de los 3 agentes en los respectivos ambientes.

entrenado con la función de recompensa TD+OA logra mejores resultados ya que alcanza el objetivo un 76 % de las veces y reduce las colisiones a solamente el 3 % de las ocasiones. Sin embargo, en entornos complejos y con mínimos locales, no es suficiente solamente con evitar obstáculos de forma eficiente. Esto se refleja en el incremento de "timeouts" respecto al agente TD (de 4 % a 21 %). Por otro lado, el agente entrenado con la función de recompensa propuesta es el único que logra alcanzar buenas performances y llega al objetivo el 96 % de las veces.

Por otro lado, para evaluar la capacidad de generalizar hacia escenarios no entrenados, se obtuvieron resultados consistentes con lo visto durante el entrenamiento. El agente entrenado con TD aprende a ser demasiado cauteloso y se mueve lejos de los obstáculos penalizando al largo trayectoria. El agente entrenado con TD+OA es menos cauteloso y aprende trayectorias más cortas pero, en el caso de escenarios con mínimos locales (como lo es el Env_2), queda atrapado y su performance no es mejor que la del agente TD. El agente entrenado con la recompensa propuesta es capaz de aprender trayectorias más cortas respecto a los otros gracias a la exploración dada por el término de entropía. Incluso en escenarios desconocidos, el agente es capaz de escapar de mínimos locales, por lo tanto logró incorporar e integrar de forma efectiva las capacidades de navegación y exploración.

Finalmente, se comparan las trayectorias generadas por el planificador RL propuesto contra las generadas por *move_base* (planificador de ROS) sobre el mismo conjunto de objetivos en el escenario Env_1. Ambos pueden alcanzar todos los objetivos planteados, sin embargo, la distancia total recorrida por el agente del enfoque propuesto es menor (14,7m vs 16,3). Además, la trayectoria generada por *move_base* no parece ser tan suave como la otra:

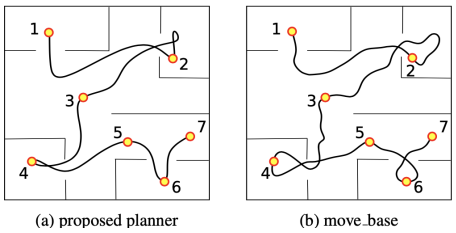


Figura 12: Comparación con algoritmo *move_base* de ROS.

2.7.3. Conclusiones

Aprendiendo mejores habilidades de exploración, comparado con una función de recompensa enfocada en el destino (TD) y una función recompensa enfocada en el destino y evitar obstáculos (TD+OA), el agente entrenado con el enfoque propuesto logra mejores resultados tanto en el escenario de entrenamiento (tasa de éxito 96 % contra 64 % y 76 % respectivamente) como en 2 escenarios desconocidos (éxito 89 % y 84 % contra 58 %, 51 % y 68 %, 62 % respectivamente). Además, el agente entrenado con el enfoque propuesto logra performances similares a las alcanzadas por *move_base*, planificador DWA que requiere el mapa del entorno.

2.8. Curiosity-driven RL agent for mapping unknown indoor environments

En [1], proponen el uso de un RL y SLAM para navegar, explorar y construir de manera eficiente mapas de entornos interiores desconocidos basados en las lecturas sensoriales a bordo del robot y la información proveniente del algoritmo SLAM, como la estimación de la pose y la completitud del mapa.

El agente, para seleccionar acciones óptimas, está capacitado para sentir curiosidad por el mundo. Este concepto se traduce en la introducción de una función de recompensa impulsada por la curiosidad que anima al agente a dirigir el robot móvil hacia áreas desconocidas e invisibles del mundo y el mapa. Esta función de recompensa es impulsada por la curiosidad dependiente de la novedad de la posición visitada por el robot que permite un rápido aprendizaje de la política de exploración y su generalización a entornos no entrenados. Además, la función de recompensa propuesta también es independiente del tipo de mapa, ya sea mapa basado en características o

basado en ubicación, construido utilizando el algoritmo SLAM.

2.8.1. Propuesta

El objetivo es resolver el problema de SLAM activo utilizando RL para seleccionar las mejores acciones para explorar los entornos interiores y construir sus mapas. Suponen que este problema tiene un horizonte finito, es decir, asumen la existencia de un estado terminal, alcanzable en un número finito de pasos, correspondiente a la exploración total del entorno y la construcción completa del mapa. El algoritmo de RL, es decir, DDPG, se basa en 80 lecturas 2D LiDAR, la estimación de la pose del robot procedente del algoritmo RBPF de SLAM, la acción anterior realizada por el agente, el porcentaje del mapa que se explorará y los pasos de tiempo restantes antes del final del episodio. Por último, en DDPG, tanto la función de política como la de valor se aproximan mediante redes neuronales.

Proponen una función de recompensa que combina una bonificación cuando el entorno está completamente explorado y el mapa está completo, una penalización cuando ocurren colisiones y una recompensa por curiosidad que promueve la exploración de lugares desconocidos. La función de recompensa propuesta, llamada Curiosidad, se muestra en la Ecuación 22.

$$R(s_t) = \begin{cases} r_{mapCompleted}, & c_t > \bar{c} \\ r_{crashed}, & z_t < z_{min} \\ r_t^c, & otherwise. \end{cases} \tag{22}$$

dónde $r_{mapCompleted}$ es la bonificación otorgada al agente cuando el porcentaje de compleción del mapa c_t está por encima de un umbral $\bar{c}$. $r_{crashed}$ es una penalización negativa por acercarse demasiado, de acuerdo con un umbral fijo z_{min}, a obstáculos según el LiDAR actual leyendo z_t y r_t^c es el término de curiosidad que promueve la exploración.

Comparan la función de recompensa (22) con las funciones de recompensa Esparza (10), Completitud de Mapa (11) y Ganancia de Información (12). En particular, se enfocan en la velocidad de aprendizaje de los agentes, entrenados con diferentes funciones de recompensa, la duración y calidad de la trayectoria después del entrenamiento, el porcentaje de mapa completado y la generalización a entornos no entrenados.

Para estudiar el efecto de las funciones de recompensa sobre las actuaciones y la habilidad de generalización de los agentes entrenan a los agentes en un solo entorno, Env-1, y evaluan sus actuaciones en dos desconocidos, Env-2 y Env-3. Los entornos de entrenamiento y evaluación se muestran en la Figura 13.

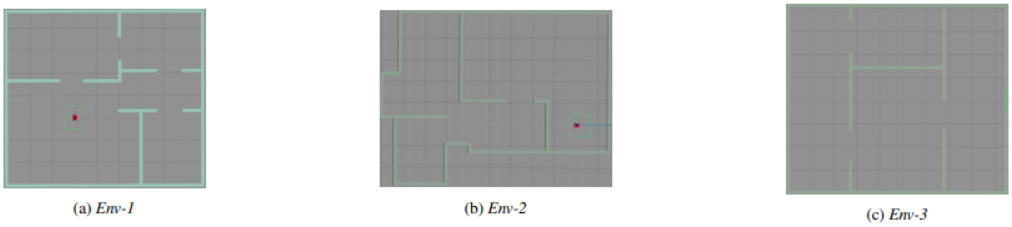


Figura 13: Ambientes de entrenamiento y evaluación.

2.8.2. Conclusiones

Cuando se elige una única posición inicial fija para el robot, en cada episodio de entrenamiento, el agente entrenado con la función de recompensa impulsada por la curiosidad (22) supera a las demás en términos de tasa de compleción del mapa, al converger rápidamente después de aproximadamente 150 episodios y el número de acciones en comparación con los otros agentes. La función de recompensa propuesta basada en la curiosidad motiva rápidamente al agente a elegir acciones que puedan conducir al robot a ubicaciones invisibles del entorno, lo más lejos posible de las conocidas. Por otro lado, cuando el robot se genera en diferentes ubicaciones de inicio, el agente entrenado con la función de recompensa Esparza (10), Completitud del Mapa (11) y Ganancia de Información (12) logran un rendimiento comparable a la curiosidad.

En comparación con los otros agentes, el que está entrenado con la función de recompensa propuesta logra la mayor completitud del mapa y viaja por caminos más suaves y cortos en comparación con los demás.

El agente impulsado por la curiosidad logra desempeños comparables a las exploraciones basadas en la frontera en términos de completitud promedio del mapa y longitud de la trayectoria. Sin embargo, vale la pena mencionar que el planificador propuesto tiene una mayor eficiencia computacional en comparación con la frontera, donde se deben realizar más operaciones en cada paso de tiempo, como detectar

las fronteras, elegir una de acuerdo con un criterio predeterminado y navegar a la frontera sin colisiones.

El elemento clave del éxito del enfoque es transformar el problema de la exploración de entornos desconocidos en el problema de visitar lugares novedosos del mundo y el mapa. Esto se hace dando forma a la función de recompensa, utilizada para entrenar al agente de RL, para alentar su curiosidad hacia lugares invisible. El agente de RL, entrenado con la función de recompensa impulsada por la curiosidad propuesta, supera en términos de generalización en entornos no entrenados, completitud del mapa y longitud y suavidad de la trayectoria, a los agentes entrenados con funciones de recompensa comúnmente utilizadas para tales tareas. El enfoque propuesto logra un rendimiento comparable al método de exploración basado en fronteras [21], pero con un menor costo de cálculo. Además, el enfoque no se limita a mapas de cuadrícula de ocupación, este es el caso de la exploración basada en fronteras, pero puede hacer frente a cualquier tipo de representación cartográfica y algoritmo SLAM. Esto se debe al hecho de que el algoritmo de entrenamiento y prueba solo requiere la estimación de la pose del robot y la integridad del mapa.

Referencias

[1] N. Botteghi y col. "CURIOSITY-DRIVEN REINFORCEMENT LEARNING AGENT FOR MAPPING UNKNOWN INDOOR ENVIRONMENTS". En: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* V-1-2021 (2021), págs. 129-136. DOI: 10.5194/isprs-annals-V-1-2021-129-2021. URL: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/V-1-2021/129/2021/.

[2] N. Botteghi y col. "Entropy-Based Exploration for Mobile Robot Navigation: A Learning-Based Approach". English. En: Planning and Robotics Workshop, PlanRob 2020, PlanRob 2020 ; Conference date: 22-10-2020 Through 23-10-2020. 2020.

[3] N. Botteghi y col. *On reward shaping for mobile robot navigation: A reinforcement learning and SLAM based approach*. English. WorkingPaper. arXiv.org, 2020.

[4] N. Botteghi y col. "Reinforcement learning helps slam: Learning to build maps". English. En: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 43.B4 (ago. de 2020). XXIVth ISPRS Congress 2020, ISPRS 2020 ; Conference date: 04-07-2020 Through 10-07-2020, págs. 329-336. ISSN: 1682-1750. DOI: 10.5194/isprs-archives-XLIII-B4-2020-329-2020. URL: http://www.isprs2020-nice.com.

[5] Ming Hsiao y col. "ARAS: Ambiguity-aware Robust Active SLAM based on Multi-hypothesis State and Map Estimations". En: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, págs. 5037-5044. DOI: 10.1109/IROS45743.2020.9341384.

[6] Julio A. Placed y José A. Castellanos. "A Deep Reinforcement Learning Approach for Active SLAM". En: *Applied Sciences* 10.23 (2020). ISSN: 2076-3417. DOI: 10.3390/app10238386. URL: https://www.mdpi.com/2076-3417/10/23/8386.

[7] Shuhuan Wen y col. "Path planning for active SLAM based on deep reinforcement learning under unknown environments". English. En: *Intelligent Service*

Robotics 13.2 (abr. de 2020). This work was supported by the National Natural Science Foundation of China under Grant No. 61773333., págs. 263-272. ISSN: 1861-2776. DOI: 10.1007/s11370-019-00310-w.

[8] F. Zeng, C. Wang y S. Ge. "A Survey on Visual Navigation for Artificial Agents With Deep Reinforcement Learning". En: *IEEE Access* 8 (2020), págs. 135426-135442. DOI: 10.1109/ACCESS.2020.3011438.

[9] Ming Hsiao y Michael Kaess. "MH-iSAM2: Multi-hypothesis iSAM using Bayes Tree and Hypo-tree". En: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, págs. 1274-1280. DOI: 10.1109/ICRA.2019.8793854.

[10] Seyed Sajad Mousavi, Michael Schukat y Enda Howley. "Deep Reinforcement Learning: An Overview". En: *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016*. Ed. por Yaxin Bi, Supriya Kapoor y Rahul Bhatia. Cham: Springer International Publishing, 2018, págs. 426-440. ISBN: 978-3-319-56991-8.

[11] Richard S. Sutton y Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[12] Cesar Cadena y col. "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age". En: *IEEE Transactions on Robotics* 32.6 (2016), págs. 1309-1332. DOI: 10.1109/TRO.2016.2624754.

[13] Timothy P. Lillicrap y col. "Continuous control with deep reinforcement learning." En: *ICLR*. Ed. por Yoshua Bengio y Yann LeCun. 2016. URL: http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#LillicrapHPHETS15.

[14] Matthew Hausknecht y Peter Stone. "Deep Recurrent Q-Learning for Partially Observable MDPs". En: (jul. de 2015).

[15] Volodymyr Mnih y col. "Playing Atari with Deep Reinforcement Learning". En: (dic. de 2013).

[16] Henry Carrillo, Ian Reid y José A. Castellanos. "On the comparison of uncertainty criteria for active SLAM". En: *2012 IEEE International Conference on Robotics and Automation*. 2012, págs. 2080-2087. DOI: 10.1109/ICRA.2012.6224890.

[17] Springer-Verlag, Berlin y Heidelberg. *Springer Handbook of Robotics*. Springer, 2008. ISBN: 978-3-319-32552-1. URL: https://www.springer.com/gp/book/9783319325507.

[18] Wolfram Burgard y Dieter Fox. "Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)". En: ene. de 2005. ISBN: 0-262-20162-3.

[19] Kevin P. Murphy. "Bayesian Map Learning in Dynamic Environments". En: *In Neural Info. Proc. Systems (NIPS*. MIT Press, 2000, págs. 1015-1021.

[20] Sebastian Thrun, Wolfram Burgard y Dieter Fox. "A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots". En: *Auton. Robots* 5.3–4 (jul. de 1998), págs. 253-271. ISSN: 0929-5593. DOI: 10.1023/A:1008806205438. URL: https://doi.org/10.1023/A:1008806205438.

[21] B. Yamauchi. "A frontier-based approach for autonomous exploration". En: *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*. 1997, págs. 146-151. DOI: 10.1109/CIRA.1997.613851.