UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

APRENDIZAJE REFORZADO DE POLÍTICAS ROBUSTAS PARA NAVEGACIÓN ROBÓTICA LOCAL

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

FRANCISCO IGNACIO LEIVA CASTRO

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTÍN

MIEMBROS DE LA COMISIÓN:
MARTIN ADAMS
EDUARDO MORALES MANZANARES

Este proyecto ha sido parcialmente financiado por
CONICYT-PFCHA/Magíster Nacional/2018-22182130, Proyecto FONDECYT 1161500 y
Proyecto FONDECYT 1201170

SANTIAGO DE CHILE
2021

RESUMEN DE LA TESIS PARA OPTAR
AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA
Y AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: FRANCISCO IGNACIO LEIVA CASTRO
FECHA: 2021
PROF. GUÍA: JAVIER RUIZ DEL SOLAR SAN MARTÍN

APRENDIZAJE REFORZADO DE POLÍTICAS ROBUSTAS PARA NAVEGACIÓN ROBÓTICA LOCAL

El problema de navegación autónoma en robótica móvil es uno de los más importantes del área, y consecuentemente, ha sido estudiado durante décadas. En esta tesis, este problema es abordado empleando aprendizaje reforzado. Considerando la descomposición de tareas que sigue la mayoría de los sistemas de navegación clásicos, el foco es puesto en dos sub-problemas específicos: evasión de colisiones, y planificación local. Si bien ambos sub-problemas han sido abordados usando aprendizaje reforzado anteriormente, en este trabajo se muestra que mejoras en desempeño, flexibilidad y aplicabilidad de las políticas entrenadas, pueden lograrse al representar las observaciones de los agentes mediante nubes de puntos 2D para la codificación de mediciones de rango. Mediante extensivas evaluaciones experimentales sobre los casos de estudio abordados, se comprueba que el desempeño final de los agentes mejora al adoptar la representación propuesta, en lugar de representaciones alternativas previamente estudiadas. También se verifica que el enfoque propuesto es compatible con el uso de representaciones diferentes para las observaciones, como imágenes y/o vectores. Finalmente, se muestra que este enfoque provee a los agentes con robustez frente a perturbaciones extremas sobre sus observaciones, y posibilita el aumento de sus capacidades perceptuales mediante estrategias de pre-procesamiento y/o fusión de datos sensoriales.

Agradecimientos

Primeramente, agradezco al profesor Javier Ruiz del Solar por guiar el trabajo presentado en esta tesis, y facilitar su desarrollo a través de productivas discusiones y consejos.

Agradezco también al profesor Martin Adams y al profesor Eduardo Morales Manzanares por las constructivas críticas que me dieron tras revisar una versión preliminar de este documento, y por formar parte de la comisión evaluadora de este trabajo.

También agradezco a Kenzo Lobos Tsunekawa, por haber colaborado en las etapas iniciales del desarrollo de esta tesis, y por haberme introducido al aprendizaje reforzado.

Agradezco también a los miembros y ex-miembros del laboratorio de robótica de la Universidad de Chile, por propiciar las enriquecedoras experiencias académicas que compartimos. En particular, agradezco a Cristopher Gómez por enseñarme a utilizar a Pepper, a Ulises Campodónico por ayudarme a acondicionar al robot Pioneer 3-DX, y a José Astorga y Rodrigo Pérez Dattari por ayudarme a grabar videos para la documentación de algunos experimentos.

De modo más personal, agradezco a mis padres por el apoyo que me han brindado durante la realización de este trabajo.

Finalmente, agradezco el financiamiento otorgado por CONICYT mediante los proyectos FONDECYT 1161500 y FONDECYT 1201170, y la beca CONICYT-PFCHA/Magíster Nacional/2018-22182130.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Motivación y definición del problema

El problema de navegación autónoma en robótica móvil es uno de los más importantes del área: prácticamente todas las aplicaciones concebibles en torno a la utilización de un robot móvil autónomo, se basan en que este cuente con la habilidad de alcanzar objetivos de navegación. La importancia del problema de navegación se ha traducido en años de investigación, y consecuentemente, en múltiples propuestas para abordarlo (e.g. [1, 2, 3, 4, 5, 6, 7]).

Un enfoque clásico, y ampliamente adoptado, consiste en descomponer el problema de navegación en diferentes sub-problemas (e.g. mapeo, localización, planificación global y planificación local). Considerando esta descomposición, la solución al problema de navegación original se obtiene a partir de la interacción entre las soluciones asociadas a cada sub-problema.

La descomposición seguida por los enfoques clásicos, por construcción, induce una modularización sobre la solución al problema de navegación. Dependiendo de las características de las soluciones asociadas a cada sub-problema, el sistema final de navegación puede contar con un gran número de parámetros. El ajuste de estos parámetros, por su parte, puede ser desafiante, y depender no solo de las soluciones a cada sub-problema y sus interacciones, sino también del ambiente en que el robot es desplegado, y de las características específicas de dicho robot.

En la última década, diversos campos de estudio han logrado importantes avances gracias al uso de metodologías basadas en aprendizaje (e.g. [8, 9, 10]). Estas metodologías son vistas como alternativas atractivas para abordar el problema de navegación robótica, dado que potencialmente podrían soslayar las limitaciones que actualmente plagan a los sistemas desarrollados siguiendo enfoques clásicos, a saber, los detrimentos en desempeño que experimentan al ser desplegados en ambientes desconocidos, y la necesidad de contar con conocimiento experto para configurar sus parámetros.

Dentro de las metodologías basadas en aprendizaje que han sido aplicadas para abordar el problema de navegación, destaca el aprendizaje reforzado. En el marco de trabajo del aprendizaje reforzado, un agente busca aprender comportamientos que le permitan lograr un cierto objetivo,

a partir de las experiencias resultantes de su interacción con el ambiente. A diferencia de metodologías alternativas, como el aprendizaje supervisado, el aprendizaje reforzado no requiere de conocimiento experto para la generación de demostraciones que permitan el aprendizaje de una tarea. Más aún, su planteamiento clásico enmarca a los problemas que aborda como problemas de toma de decisiones secuenciales, por lo que su aplicación resulta natural al modelar una amplia gama de problemas en robótica, en particular, el de navegación.

Si bien la aplicación del aprendizaje reforzado ha sido exitosa en múltiples casos de estudio, logrando resultados sin precedentes en varios problemas (e.g. [11, 8, 12, 10, 13, 14, 15, 16]), la adopción de esta metodología en robótica conlleva una serie de desafíos. La cantidad necesaria de interacciones agente-ambiente para lograr aprender comportamientos deseables es a menudo muy grande. Estas interacciones, además, pueden poner al sistema físico (y a su entorno) en riesgo, considerando que la ejecución de acciones exploratorias es necesaria para el aprendizaje. Al ser la conducción del entrenamiento en el mundo real comúnmente impracticable, es habitual que simuladores sean empleados con este propósito.

El uso de simuladores permite acelerar la generación de experiencias para el aprendizaje, no obstante, las diferencias entre las interacciones simuladas y reales dan lugar a un nuevo problema, conocido como el problema del "*reality-gap*". La brecha simulación-realidad, o *reality-gap*, se manifiesta por ejemplo, cuando la dinámica de las interacciones simuladas difiere excesivamente de la dinámica de las interacciones reales, o cuando esta clase de diferencias se presentan en las observaciones del ambiente que el agente recibe. Cuando esta brecha es demasiado prominente, el desempeño de un agente entrenado en simulaciones se ve afectado fuertemente al ser desplegado en el mundo real. En casos extremos, esto puede implicar que su comportamiento sea completamente diferente al que muestra en simulaciones.

Con el fin de aliviar las dificultades anteriormente descritas, diversos métodos han sido propuestos: desde la utilización de simuladores más realistas o de técnicas basadas en la aleatorización de alguna de sus propiedades [17, 18, 19, 20], hasta el uso de modelos generativos [21], o la incorporación de datos reales al entrenamiento. Los esfuerzos por facilitar la transferencia de agentes entrenados usando aprendizaje reforzado al mundo real, muestran el interés de la comunidad científica por lograr la extrapolación de los excelentes resultados que esta metodología ha logrado en otros casos de estudio, a aplicaciones en robótica.

En este contexto, en esta tesis se aborda el problema de navegación robótica usando aprendizaje reforzado. Si bien usando esta metodología el problema puede ser tratado de manera "*end-to-end*", a la fecha, no existe evidencia de que este enfoque haya logrado la creación de un sistema que sea desplegable en el mundo real y presente un muy buen desempeño. Considerando esto, la descomposición de tareas que siguen los sistemas de navegación clásicos es adoptada, y el foco es puesto en dos sub-problemas específicos: la evasión de colisiones, y la planificación local.

Si bien ambos problemas ya han sido abordados usando aprendizaje reforzado en el pasado (e.g. [22, 23, 24]), algunas decisiones de diseño claves para el despliegue de esta clase de sistemas en el mundo real, como la codificación de las observaciones del agente, o la parametrización de su política (o controlador), no han sido discutidas en profundidad. Con este trabajo se busca contribuir al cierre de esta brecha en la literatura, no solo analizando el impacto que esta clase de decisiones de diseño genera, sino también demostrando que mejoras en el desempeño y la aplicabilidad de las políticas entrenadas pueden ser logradas al estudiarlas en mayor detalle.

En concreto, y considerando la amplia adopción de sensores de rango como fuentes de información para robots móviles, en esta tesis se propone el uso de nubes de puntos 2D de tamaño variable para la representación de las observaciones de los agentes entrenados usando aprendizaje reforzado, y la utilización de extractores de características *ad-hoc* como parte de la parametrización de sus políticas.

Mediante el desarrollo de los casos de estudio que son presentados en este trabajo, se prueban los beneficios del enfoque propuesto a través de experimentos conducidos en simulaciones, y en el mundo real. En primer lugar, se prueba que un mejoramiento en el desempeño final de los agentes puede ser logrado al emplear la representación propuesta, en reemplazo de codificaciones alternativas para las observaciones que han sido exploradas previamente. En segundo lugar, se prueba que el enfoque es flexible, y compatible con el uso de fuentes de información diferentes, como imágenes. Finalmente, se prueba que este enfoque provee a los agentes entrenados con robustez a perturbaciones extremas sobre sus observaciones, y posibilita el uso de estrategias basadas en pre-procesamiento y/o fusión de datos sensoriales aplicadas "sobre la marcha", para mejorar las capacidades perceptuales de los agentes.

1.2. Hipótesis

En el contexto del aprendizaje reforzado de políticas para navegación robótica, la representación de observaciones mediante nubes de puntos 2D para codificar mediciones de rango puede traer importantes beneficios. El uso de esta clase de representaciones puede facilitar el aprendizaje de características invariantes al sensor utilizado, al inducir un desacoplamiento entre sus propiedades (campo de visión y resolución), y su mediciones. Como consecuencia, las políticas entrenadas siguiendo este enfoque pueden ser robustas a perturbaciones extremas sobre sus observaciones, lo que puede facilitar su despliegue en el mundo real.

1.3. Objetivos

1.3.1. Objetivo general

El objetivo de esta tesis consiste en diseñar, implementar y validar soluciones basadas en aprendizaje reforzado a los problemas de evasión de colisiones y planificación local para robots móviles. Considerando la hipótesis de este trabajo, las soluciones generadas emplearán nubes de puntos 2D como observaciones para los agentes entrenados. Más aún, estas soluciones deberán ser desplegables en el mundo real, por lo que los supuestos que sean considerados para su desarrollo, en particular, aquellos referentes a las capacidades perceptuales y de cómputo con las que típicamente cuentan los robots móviles, deberán apegarse a la realidad.

1.3.2. Objetivos específicos

Con el propósito de cumplir el objetivo general antes enunciado, y verificar la hipótesis de este trabajo, los siguientes objetivos específicos son considerados:

- Diseñar e implementar un sistema de evasión de colisiones basado en aprendizaje reforzado, que emplee nubes de puntos 2D como parte de las observaciones del agente.
- Extender el sistema de evasión de colisiones desarrollado para crear uno que resuelva el problema de planificación local, nuevamente empleando aprendizaje reforzado y nubes de puntos 2D como parte de las observaciones del agente.
- Validar experimentalmente que los sistemas desarrollados (para el problema de evasión de colisiones y el problema de planificación local), puedan ser desplegados en el mundo real.

1.4. Alcances de este trabajo

El desarrollo y validación experimental del sistema de evasión de colisiones es realizado en torno al robot Pepper. Análogamente, el desarrollo y validación del sistema de planificación local es realizado en torno al robot Pioneer 3-DX. La aplicabilidad de estos sistemas en plataformas robóticas diferentes, consecuentemente, podría requerir la introducción de cambios sobre las formulaciones subyacentes a cada sistema. Estos cambios, no obstante, tienen relación con la plataforma de despliegue objetivo (e.g. su geometría y sensorización), mas no con aspectos fundamentales de las propuestas que son planteadas en este trabajo, los cuales son de carácter general.

1.5. Estructura del documento

El presente documento se estructura como sigue:

- En el Capítulo 2 se presentan los antecedentes generales necesarios para la comprensión de lo expuesto en los capítulos subsecuentes. En particular, se introduce una breve revisión del aprendizaje reforzado, y del aprendizaje reforzado profundo y la aproximación de funciones mediante el uso de redes neuronales artificiales.
- En el Capítulo 3 se presenta el primer caso de estudio abordado: el desarrollo de un sistema de evasión de colisiones mediante aprendizaje reforzado multimodal para un robot de servicio con capacidades perceptuales limitadas. Los contenidos de este capítulo han sido previamente publicados en [25].
- En el Capítulo 4 se presenta el diseño e implementación de un planificador local para navegación robótica, basado en aprendizaje reforzado, que emplea nubes de puntos 2D como observaciones. Los expuesto en este capítulo ha sido previamente publicado en [26].
- Finalmente, en el Capítulo 5 se presentan las conclusiones derivadas del trabajo realizado, y las posibles direcciones futuras en que este puede ser extendido.

Capítulo 2

Antecedentes

2.1. Procesos de Decisión de Markov

Los Procesos de Decisión de Markov (*Markov Decision Processes* [MDPs]) [27] son una formalización matemática clásica empleada para modelar problemas de toma de decisiones secuenciales. En particular, los MDPs son útiles para formalizar el problema de aprender una cierta tarea a partir de las interacciones entre un "agente" (quien aprende y toma las decisiones) y su "ambiente" (todo aquello con lo que el agente interactúa; el entorno en el que está inmerso).

Formalmente, un MDP es definido por la tupla $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r)$, donde:

- $\mathcal{S}$ es un conjunto de estados.
- $\mathcal{A}$ es un conjunto de acciones.
- $\mathcal{T}(s, a, s')$ es la función de transición de estados que determina la dinámica de la interacción agente-ambiente. Entrega la probabilidad de que el ambiente transicione al estado s' dado que se encontraba en el estado s y la acción a fue ejecutada, esto es, $\mathcal{T}(s, a, s') = p(s_{t+1} = s' | s_t = s, a_t = a)$.
- $r(s, a)$ es la función de recompensa. Especifica la recompensa esperada para cada transición de estado, es decir, $r(s, a) = \mathbb{E}[r_t | s_t = s, a_t = a]$.

En un proceso de decisión de Markov, el agente y el ambiente interactúan en pasos de tiempo discretos $t \in \{0, 1, 2, ..., T\}$. En cada paso de tiempo t, el agente recibe el estado del ambiente, s_t, y en función de éste, ejecuta una acción a_t. La ejecución de esta acción causa que el ambiente evolucione a un nuevo estado s_{t+1}, de acuerdo a $\mathcal{T}(s, a, s')$, y que el agente reciba una recompensa r_t, de acuerdo a $r(s, a)$.

Bajo esta formulación, tanto la probabilidad de llegar a un estado s_{t+1} como la de obtener una recompensa r_t, quedan completamente determinadas por el par estado-acción (s_t, a_t). Esto se conoce como la "propiedad de Markov" [28]. La Figura 2.1 muestra una representación gráfica de la interacción agente-ambiente que ha sido descrita.

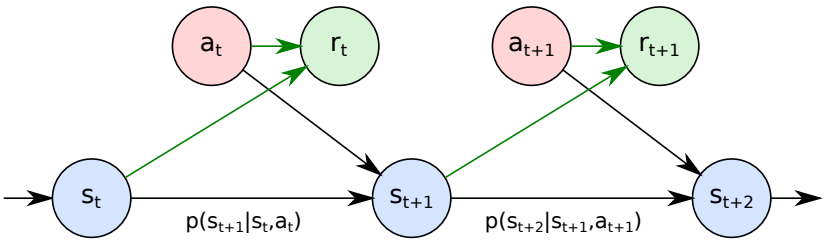


Figura 2.1: Diagrama de la interacción agente-ambiente definida en un MDP.

2.1.1. Observabilidad parcial

Cuando el agente solo tiene acceso a observaciones de los estados del ambiente, siendo estas observaciones representaciones incompletas y/o ruidosas de dichos estados, se dice que el MDP es parcialmente observable (*Partially Observable Markov Decision Processes* [POMDPs]). Formalmente, un POMDP es definido por la tupla $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \Omega, \mathcal{O})$, donde:

- $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r)$ define un MDP.
- Ω es un conjunto de observaciones.
- $\mathcal{O}(s', a, o)$ es la función de observación. Para cada acción a y estado siguiente s', especifica la probabilidad de que el agente reciba a o como observación, es decir, $\mathcal{O}(s', a, o) = p(o_{t+1} = o|s_{t+1} = s', a_t = a)$.

En este caso, en cada instante de tiempo el agente recibe una observación o_t de acuerdo a $O(s', a, o)$, la cual emplea para decidir y ejecutar una acción a_t. La evolución del estado del ambiente y la generación de la recompensa que recibe el agente sigue las mismas reglas que en el caso completamente observable. La Figura 2.2 ilustra este proceso.

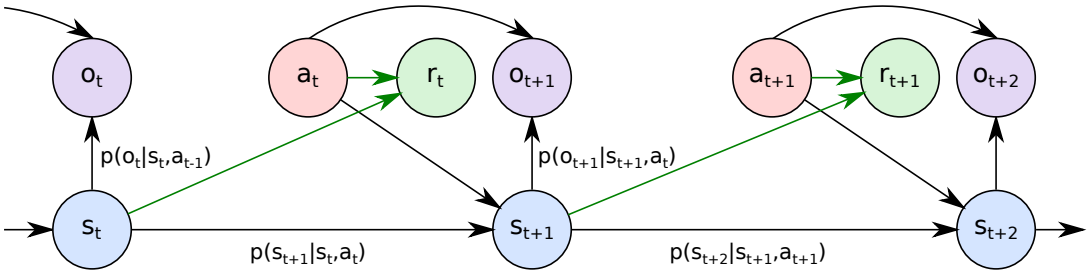


Figura 2.2: Diagrama de la interacción agente-ambiente definida en un POMDP.

2.2. Aprendizaje reforzado

El aprendizaje reforzado (*reinforcement learning* [RL]) es simultáneamente un problema, un conjunto de soluciones a dicho problema, y una disciplina científica que estudia tanto al problema como a sus soluciones [28]. Más precisamente, el aprendizaje reforzado es uno de los tres principales sub-campos del aprendizaje de máquinas, junto al aprendizaje supervisado y al aprendizaje no supervisado.

El aprendizaje reforzado provee un marco de trabajo para la resolución de problemas de toma de decisiones secuenciales. Su objetivo es lograr que un agente aprenda a realizar una determinada tarea a partir de la experiencia que adquiere al interactuar con el ambiente. En su versión estándar, la interacción entre el agente y el ambiente es modelada como un proceso de decisión de Markov, donde el agente ejecuta acciones en función de los estados que visita, de acuerdo a una política $\pi(a|s)$. La interacción agente-ambiente se desarrolla como fue descrita en la Sección 2.1, esto es, el agente recibe un estado s_t, ejecuta una acción a_t según $\pi(a_t|s_t)$, y esto hace que simultáneamente el ambiente transicione a un nuevo estado s_{t+1}, y que el agente reciba una recompensa r_t.

El objetivo del agente es lograr aprender un mapeo entre acciones y estados que permita que la recompensa que acumula a lo largo de su interacción con el ambiente se maximice. Formalmente, se busca maximizar el valor esperado del "retorno" R_t, siendo este último definido en términos de la secuencia de recompensas que el agente recibe a partir de un instante de tiempo t.

La definición del retorno depende de las características del problema a resolver. Cuando la interacción agente-ambiente tiene estados terminales bien definidos, esta se dice de carácter "episódico". En esta clase de problemas, la interacción entre el agente y el ambiente puede ser dividida en "episodios", cada uno de ellos iniciando y terminando en un cierto subconjunto de estados posibles. Esta característica hace que la interacción agente-ambiente este acotada por un horizonte temporal $T < \infty$, por lo que el retorno puede ser definido como la suma de las recompensas obtenidas por el agente a partir de un cierto paso de tiempo: $R_t = \sum_{k=t}^{T} r_k$.

Por el contrario, en algunos problemas no existen estados terminales, por lo que la interacción agente-ambiente no está acotada temporalmente ($T = \infty$). En estos casos la definición de R_t para el caso episódico podría diverger, por lo que se introduce un factor de descuento $\gamma \in [0, 1]$ que pondera a las recompensas instantáneas en función del paso de tiempo en que son recibidas. Este factor da lugar a un retorno descontado, el cual se define como $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$.

La introducción de γ cumple dos propósitos:

1. Permite una definición apropiada para el retorno cuando $T = \infty$, suponiendo que las recompensas r_t están acotadas, y que $\gamma < 1$.
2. Permite otorgar importancias relativas a las recompensas instantáneas, en función del paso de tiempo en que son obtenidas. Un factor de descuento cercano a cero fomentará el aprendizaje de comportamientos que maximicen recompensas inmediatas, mientras que un factor de descuento cercano a uno fomentará el aprendizaje de comportamientos que otorgan mayor relevancia a las recompensas futuras.

Para terminar de formalizar el objetivo del aprendizaje reforzado, es necesario considerar la naturaleza estocástica del ambiente. La interacción agente-ambiente da lugar a una distribución de probabilidades sobre las secuencias de los pares estado-acción (en adelante, "trayectorias") que son visitadas por el agente. Si el estado inicial que el agente visita está determinado por una distribución de probabilidades $p(s)$, y los pasos de tiempo en que el agente interactúa con el ambiente están acotados por T, entonces la distribución sobre las trayectorias recorridas por el agente está dada por (2.1).

$$p_\pi(s_1, a_1, ..., s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{2.1}$$

El objetivo del aprendizaje reforzado consiste en maximizar el retorno obtenido por el agente a lo largo de las trayectorias que recorre en su interacción con el ambiente. De este modo, denotando $p_\pi(\tau) = p_\pi(s_1, a_1, ..., s_T, a_T)$, el objetivo formal del aprendizaje reforzado consiste en la maximización de la función definida en (2.2).

$$J_{\text{RL}}(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t) \right] \tag{2.2}$$

2.2.1. Funciones de valor

En términos prácticos, la resolución de un problema de aprendizaje reforzado se logra encontrando una política $\pi(a|s)$ que maximice la función objetivo definida en (2.2). Para encontrar dicha política, es necesario considerar los retornos futuros que el agente recibirá en función de las acciones que ejecuta en cada paso de tiempo. Para formalizar esta noción, dos funciones comúnmente utilizadas en la mayoría de los algoritmos de aprendizaje reforzado deben ser introducidas.

La función de valor $V^\pi(s)$ entrega el valor esperado del retorno que recibirá el agente si este parte de un estado s, y rige su comportamiento siguiendo una política π. La definición de $V^\pi(s)$ está dada por (2.3), donde $\tau_t = (s_t, a_t, ...s_T, a_T)$ es la trayectoria recorrida por el agente a partir del estado s_t.

$$V^\pi(s) = \mathbb{E}_{\tau_t \sim p_\pi(\tau_t)} \left[\sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k) \middle| s_t = s \right] \tag{2.3}$$

Análogamente, la función de valor del par estado-acción $Q^\pi(s, a)$ entrega el valor esperado del retorno que recibirá el agente si este parte de un estado s, ejecuta una acción a, y luego su comportamiento se rige por una política π. Formalmente, la definición de $Q^\pi(s, a)$ está dada por (2.4).

$$Q^\pi(s, a) = \mathbb{E}_{\tau_t \sim p_\pi(\tau_t)} \left[\sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k) \middle| s_t = s, a_t = a \right] \tag{2.4}$$

Por definición, las funciones $V^\pi(s)$ y $Q^\pi(s, a)$ están mutuamente relacionadas. La función de valor $V^\pi(s)$ entrega el retorno esperado a partir de un determinado estado, considerando todas las posibles acciones que podrían ser ejecutadas por el agente siguiendo su política. Esto permite que $V^\pi(s)$ pueda ser escrita en términos de $Q^\pi(s, a)$, de acuerdo a (2.5).

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(a|s)} Q^\pi(s, a) \tag{2.5}$$

Por otro lado, $Q^\pi(s, a)$ entrega el retorno esperado tras la ejecución de una acción en un cierto estado. Si bien esta primera acción es dada, todas las acciones futuras que ejecute el agente dependen de π. Lo anterior permite que $Q^\pi(s, a)$ pueda ser escrita en términos de $V^\pi(s)$, de acuerdo a (2.6).

$$Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} V^\pi(s') \tag{2.6}$$

Usando (2.5) y (2.6), es posible establecer relaciones de recurrencia para $V^{\pi}(s)$ y $Q^{\pi}(s,a)$. Estas relaciones son denominadas "ecuaciones de Bellman", y permiten la estimación de cada función a través de una variedad de métodos. Reemplazando (2.6) en (2.5), la ecuación de Bellman para $V^{\pi}(s)$ queda dada por (2.7).

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(a|s)}\left[r(s,a) + \gamma \mathbb{E}_{s'\sim p(s'|s,a)} V^{\pi}(s')\right] \tag{2.7}$$

Análogamente, la ecuación de Bellman para $Q^{\pi}(s,a)$ queda dada por (2.8).

$$Q^{\pi}(s,a) = r(s,a) + \gamma \mathbb{E}_{s'\sim p(s'|s,a)} \mathbb{E}_{a'\sim\pi(a'|s')} Q^{\pi}(s',a') \tag{2.8}$$

La función $V^{\pi}(s)$ establece un orden parcial sobre las políticas: una política π es mejor que otra política π', si y solo si $V^{\pi}(s) \geq V^{\pi'}(s)$, para todo $s \in \mathcal{S}$. Para resolver el problema de optimización asociado a la maximización de $J_{\text{RL}}(\pi)$, es necesario encontrar una política óptima π^*, tal que $\pi^* = \arg\max_{\pi} J_{\text{RL}}(\pi)$.

Notando que $J_{\text{RL}}(\pi) = \mathbb{E}_{s_1\sim p(s)} V^{\pi}(s_1)$, la política óptima será aquella cuya función de valor entregue el mayor retorno esperado para cualquier estado visitable por el agente. Dicha función de valor se define de acuerdo a (2.9).

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.9}$$

La función $V^*(s)$ es la función de valor óptima, y es compartida por toda política óptima. Análogamente, toda política óptima comparte una función de valor óptima para el par estado-acción, la cual está dada por (2.10).

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{2.10}$$

Al ser funciones de valor, tanto $V^*(s)$ como $Q^*(s,a)$ cumplen con la relación de recurrencia que establece su respectiva ecuación de Bellman. Estas relaciones, no obstante, pueden ser escritas de manera especial considerando la definición de las funciones de valor óptimas. Para (2.7), el valor óptimo de $V^{\pi}(s)$ se obtiene si la acción elegida para $s_t = s$ maximiza el retorno esperado. Con esto, la ecuación de Bellman para $V^*(s)$ está dada por (2.11).

$$V^*(s) = \max_{a\in\mathcal{A}} \left[r(s,a) + \gamma \mathbb{E}_{s'\sim p(s'|s,a)} V^*(s')\right] \tag{2.11}$$

Utilizando un argumento similar, la ecuación de Bellman para $Q^*(s,a)$ puede ser escrita según (2.12).

$$Q^*(s,a) = r(s,a) + \gamma \max_{a'\in\mathcal{A}} \mathbb{E}_{s'\sim p(s'|s,a)} Q^*(s',a') \tag{2.12}$$

2.2.2. Algoritmos de aprendizaje reforzado

Los algoritmos de aprendizaje reforzado tratan de encontrar una política óptima bajo las restricciones que impone un problema de aprendizaje reforzado, y siguiendo su filosofía de aprendizaje.

Más precisamente, buscan encontrar un mapeo entre estados y acciones para que el agente maximice la recompensa que obtiene en su interacción con el ambiente, utilizando esta interacción y las recompensas obtenidas como experiencia para el aprendizaje de comportamientos deseables.

En la práctica, el mapeo entre estados y acciones que define a la política puede ser representado de muchas formas. Si el conjunto $\mathcal{S}$ de estados es pequeño, la política puede ser representada como un arreglo de tamaño $|\mathcal{S}|$, donde cada elemento del arreglo establece una relación entre un determinado estado, y la acción que debería ser ejecutada cuando el agente se encuentre en dicho estado. No obstante, en la gran mayoría de problemas que pueden ser formulados como problemas de aprendizaje reforzado, la cardinalidad de $\mathcal{S}$ es demasiado grande como para optar por esta representación. En estos problemas, la política es aproximada mediante una función parametrizada $\pi_\theta(s|a) = f(s|a;\theta)$, donde θ denota al vector de parámetros del aproximador. Considerando este último caso, el problema original se convierte en encontrar los parámetros θ^* que permitan maximizar (2.2).

$$\theta^* = \arg\max_\theta J_{\text{RL}}(\pi_\theta) \tag{2.13}$$

El vector θ^* puede ser encontrado empleando una variedad de métodos diferentes, lo que establece una taxonomía sobre los algoritmos de aprendizaje reforzado. En términos generales, existen dos formas de clasificación para los algoritmos: dependiendo de si utilizan o no un modelo de transición del ambiente, y dependiendo de qué es lo que aprenden al momento de derivar una política.

Para la primera forma de clasificación, los algoritmos que se basan en la utilización de un modelo del ambiente se denominan *Model-Based*, mientras que los que no, se denominan *Model-Free*. El modelo empleado por los algoritmos de tipo *Model-Based* puede permitir, por ejemplo, la predicción de estados y recompensas futuras, lo cual, a su vez, permite la derivación de una política mediante la planificación de acciones óptimas. En general, no obstante, el modelo debe ser aprendido a partir de la interacción agente-ambiente, proceso que puede ser muy complejo, y cuyo éxito, dependiendo del enfoque adoptado, podría depender de la disponibilidad de vasto conocimiento experto del problema a resolver.

En contraste, los algoritmos de tipo *Model-Free* no emplean un modelo de la dinámica del ambiente, ni tienen por objetivo aprenderlo. Estos algoritmos tratan de aprender una política a partir de la interacción agente-ambiente, sin consideraciones relacionadas a la dinámica de dicha interacción. A diferencia de los algoritmos de tipo *Model-Based*, en general no requieren de un vasto conocimiento experto del problema a resolver, pero usualmente sí de una mayor cantidad de interacciones entre el agente y el ambiente para aprender una política.

Para la segunda forma de clasificación, en general se distinguen tres familias de algoritmos. La primera familia agrupa a métodos que tratan de encontrar directamente los parámetros de una política $\pi_\theta(a|s)$, a través de la optimización de $J_{\text{RL}}(\pi_\theta)$. Estos métodos asumen que la política es diferenciable con respecto a sus parámetros, con lo que los actualizan mediante gradiente ascendente, en la dirección indicada por $\nabla_\theta J_{\text{RL}}(\pi_\theta)$.

La segunda familia de algoritmos se caracteriza por emplear la parametrización de alguna función de valor, $V^\pi_\theta(s)$ o $Q^\pi_\theta(s,a)$, y usar esta parametrización para aproximar a la función de valor óptima correspondiente, $V^*(s)$ o $Q^*(s,a)$. Para lograr encontrar los parámetros θ^* que logran dicha aproximación, estos métodos tratan de optimizar una función estrechamente relacionada a las

ecuaciones de Bellman para $V^*(s)$ (2.11) o $Q^*(s,a)$ (2.12), dependiendo de la función de valor que se esté aproximando. El objetivo final de estos algoritmos, es emplear la aproximación de la función de valor encontrada para derivar una política para el agente, sin mantener una parametrización explícita de dicha política. A modo de ejemplo, si la función $Q^*(s,a)$ es aproximada por $Q^\pi_{\theta^*}(s,a)$, entonces $\pi(a|s)$ puede ser derivada a partir de la relación:

$$\pi(a|s) = \begin{cases} 1 & \text{si } a = \arg\max_{a \in \mathcal{A}} Q^\pi_{\theta^*}(s,a), \\ 0 & \text{si no.} \end{cases}$$

Finalmente, la tercera familia de algoritmos emplea ambos enfoques, manteniendo parametrizaciones explícitas para la política y alguna función de valor, y empleandolas conjuntamente para optimizar $J_{\text{RL}}(\pi)$. Estos algoritmos se dicen de tipo "Actor-Crítico", refiriéndose a la política como "actor" (dado que determina las acciones del agente), y a la función de valor como "crítico" (puesto que provee una estimación de "valor", en función de un cierto retorno esperado).

Una distinción adicional, a menudo empleada para caracterizar a un algoritmo, es especificar si este es "*on-policy*" u "*off-policy*". Los algoritmos *on-policy* corresponden a algoritmos que necesitan utilizar experiencias nuevas cada vez que la política es actualizada (sea esta representada de forma explítica o implícita), siendo estas experiencias generadas utilizando la política que está siendo aprendida. En contraste, los algoritmos *off-policy* pueden utilizar muestras que han sido generadas a partir de interacciones en las que el agente sigue una política arbitraria. En general, los algoritmos de tipo *off-policy* requieren una menor cantidad de experiencias para lograr el aprendizaje de una política, por lo que suelen ser más eficientes que los de tipo *on-policy* en lo referente a este punto.

Las categorías señaladas, a menudo no son suficientes para encasillar completamente a un algoritmo específico, pero permiten tener una idea general de la filosofía de aprendizaje adoptada por un método. La naturaleza y número de estas categorías, además, da cuenta de la heterogeneidad y variedad de algoritmos de aprendizaje reforzado que existen. Este último aspecto hace que usualmente la resolución de un problema de aprendizaje reforzado tenga como etapa preliminar la selección de un algoritmo, tomando en consideración sus limitaciones, fortalezas, y debilidades. Como ejemplo, algunas de las características que son consideradas al momento de escoger un algoritmo, son su compatibilidad con estados y/o acciones en espacios continuos o discretos, su eficiencia, su estabilidad, y la complejidad de su implementación.

2.3. Aprendizaje reforzado profundo

Una de las principales desventajas del aprendizaje reforzado (en su formulación clásica), es su limitada aplicabilidad a problemas con espacios de estados y acciones de alta dimensionalidad. Esta limitación, no obstante, ha sido soslayada recientemente gracias a la integración de componentes fundamentales del aprendizaje profundo (*deep learning*) al marco de trabajo del aprendizaje reforzado. Esta combinación ha dado lugar al "Aprendizaje Reforzado profundo" (*Deep Reinforcement Learning* [DRL]), campo de estudio que ha permitido la resolución de complejos problemas de toma de decisiones, antes considerados intratables.

El aprendizaje reforzado profundo se caracteriza por la utilización de redes neuronales profundas como aproximadores funcionales y/o componentes esenciales para posibilitar el manejo de espacios de alta dimensionalidad. Dependiendo del algoritmo, estas redes pueden ser utilizadas como parametrizaciones de un modelo del ambiente, de una política, de una función de valor, e incluso simplemente como preprocesadoras de información.

El primer trabajo que popularizó la utilización de redes neuronales profundas en aprendizaje reforzado fue desarrollado por Mnih et al. [29, 8]. En este trabajo, la combinación de Q-Learning [30] y redes neuronales convolucionales, permitió el entrenamiento de políticas que alcanzaron desempeños de nivel humano en una variedad de juegos de Atari 2600. Tras este primer éxito, el potencial de esta metodología fue nuevamente demostrado mediante el desarrollo de *Alpha Go* [12] y *Alpha Go Zero* [10], programas que permitieron por primera vez derrotar –por un amplio margen– a jugadores profesionales de Go. Recientemente, dos nuevos trabajos lograron resultados sin precedentes, al desarrollar agentes que superaron el desempeño de humanos en videojuegos competitivos, caracterizados por presentar observabilidad parcial y una alta complejidad. *OpenAI Five* [15] logró derrotar a los campeones mundiales de Dota II, convirtiendose en el primer sistema en vencer a los campeones de un *e-sport*. *Alpha Star* [14], por su parte, logró alcanzar un nivel de juego competitivo de grado *GrandMaster* en StarCraft II, superando de este modo al 99.8 % de los jugadores humanos oficialmente clasificados en dicho juego.

La aplicabilidad del aprendizaje reforzado profundo no se restringe al ámbito de los juegos de video. Numerosos casos de estudio han demostrado que agentes entrenados usando DRL pueden emplearse para resolver complejos problemas en un amplio espectro de campos de estudio, incluyendo robótica [31], visión computacional, procesamiento de lenguaje natural, y finanzas [32, 33].

El éxito del aprendizaje reforzado profundo se debe, principalmente, a que explota las capacidades de representación de las redes neuronales profundas, permitiendo el aprendizaje de características compactas para la resolución de problemas con espacios de estados (u observaciones) y acciones de alta dimensionalidad. En este ámbito, el aprendizaje reforzado se ha nutrido de una serie de avances que han empujado adelante el estado del arte del aprendizaje profundo: nuevas arquitecturas y componentes para redes neuronales, y una serie de consideraciones técnicas para facilitar su entrenamiento.

En las sub-secciones siguientes se presenta una breve revisión de las redes neuronales como aproximadores funcionales, y como estas se han integrado al aprendizaje reforzado. Adicionalmente, se presenta una descripción completa de dos de los trabajos más influyentes de aprendizaje reforzado profundo: *Deep Q-Network* (DQN) [29, 8] y *Deep Deterministic Policy Gradient* (DDPG) [34].

2.3.1. Aproximación funcional y redes neuronales artificiales

La utilización de redes neuronales artificiales en aprendizaje reforzado no es una tendencia nueva: muchos trabajos desarrollados antes de la popularización del aprendizaje profundo emplearon redes neuronales como aproximadores funcionales. Los resultados obtenidos en estos trabajos, no obstante, dependendían fuertemente del cuidadoso diseño de características para la representación de los estados u observaciones entregados al agente. Estas características mantenían una dimen-

sionalidad que puede considerarse baja para los estándares actuales, y eran diseñadas de forma heurística, por lo que su fabricación requería de un conocimiento experto del problema a resolver. No fue sino hasta el desarrollo de DQN [29, 8] que se demostró empíricamente que las redes neuronales profundas, bajo ciertas consideraciones, podían aprender estas características de forma implícita a partir de estados u observaciones en formatos más "crudos" (e.g. imágenes RGB).

Si bien actualmente existe una gran variedad de modelos que son considerados como componentes esenciales del aprendizaje profundo, a continuación solo se describirán las redes *feed-forward*, las redes convolucionales, y las redes recurrentes. Estos modelos han sido fundamentales para la implementación de muchas soluciones a problemas formulados utilizando DRL y, en particular, para la implementación de los casos de estudio que son expuestos en los Capítulos 3 y 4 del presente documento. Con este mismo propósito, la red neuronal PointNet [35] también es descrita a grandes rasgos.

Redes neuronales *feed-forward*

Las redes neuronales *feed-forward* son aproximadores funcionales que definen un mapeo de la forma $y = f(x;\theta) = f_\theta(x)$, donde θ es el vector de parámetros de la red. Se denominan "*feed-forward*" debido a que la información en ellas fluye de forma unidireccional, desde la entrada x a la salida y, sin que exista ninguna conexión de retroalimentación [36].

Una red neuronal *feed-forward* consiste en la composición de múltiples funciones, las cuales son denominadas "capas". Formalmente, para una red compuesta por k capas f_i con parámetros θ_i, la relación entre la entrada y la salida estará dada por (2.14), donde $\theta = [\theta_1, ..., \theta_k]$.

$$y = f_\theta(x) = (f_k \circ f_{k-1} \circ ... \circ f_1)(x) = f_k(f_{k-1}(...(f_1(x))...)) \tag{2.14}$$

La última capa de la red, f_k, se denomina "capa de salida". Cada una del resto de las capas se denomina "capa oculta". Cada capa de la red recibe un vector de entrada, y entrega un vector de salida. Si $x \in \mathbb{R}^{l_0}$ y $x_i \in \mathbb{R}^{l_i}$ corresponde a la salida de la capa i-ésima, entonces para una red de k capas, la salida $y = x_k \in \mathbb{R}^{l_k}$ puede ser calculada siguiendo las reglas definidas por (2.15) y (2.16), donde $W_i \in \mathbb{R}^{l_i \times l_{i-1}}$ y $b_i \in \mathbb{R}^{l_i}$ definen una transformación lineal, y σ_i corresponde a una "función de activación", típicamente no lineal, aplicada elemento a elemento. Los parámetros de cada capa, por tanto, están dados por los elementos que componen a W_i y a b_i, y en caso de utilizar una función de activación paramétrica, por los parámetros de σ_i.

$$x_0 = x \tag{2.15}$$

$$x_i = f_i(x_{i-1}) = \sigma_i(W_i x_{i-1} + b_i), \quad i \in \{1, ..., k\} \tag{2.16}$$

Las funciones de activación comúnmente utilizadas para las capas ocultas incluyen a la función sigmoidal, tangente hiperbólica, y ReLU. En contraste, la función de activación de la capa de salida, σ_k, típicamente depende de la tarea que deba realizar la red neuronal, y de la forma en que esta tarea haya sido modelada.

Las redes neuronales deben su nombre a la analogía que puede establecerse entre su estructura y algunas propiedades biológicas de las neuronas y sus conexiones [36]. En lugar de pensar en la red

neuronal *feed-forward* como la composición de una serie de funciones, cada uno de los elementos de salida de dichas funciones puede ser interpretado como una "unidad" que se comporta como una neurona: las unidades de una capa dada, entregan una señal de respuesta al estímulo proveniente de las unidades de la capa anterior (o de la entrada, en el caso de la primera capa) [1].

Las redes neuronales son entrenadas con el fin de aproximar un cierto mapeo $y = f^*(x)$. Para lograr este fin, los parámetros $\theta = [\theta_1, ..., \theta_k]$ que determinan el comportamiento de la red son ajustados utilizando algún método de aprendizaje. El más popular de estos métodos corresponde a la utilización conjunta del algoritmo "*Back-Propagation*" [37], y "*Stochastic Gradient Descent*" (SGD) o alguna de sus variantes.

Redes neuronales convolucionales

Las redes neuronales convolucionales son redes *feed-forward* que emplean la operación de convolución en al menos una de sus capas. Estas redes son especialmente útiles para procesar entradas que presentan una estructura conocida, por ejemplo, imágenes [36].

En general, las redes neuronales convolucionales emplean variaciones multidimensionales de la operación de convolución discreta definida en (2.17), donde x es la entrada, k es un filtro o *kernel* de convolución con una región de soporte definida por el conjunto $\{-M, ..., M\}$, e y es la salida.

$$y(t) = (k * x)(t) = \sum_{m=-M}^{M} x(t-m)k(m) \tag{2.17}$$

Esta operación y sus variaciones multidimensionales, al ser lineales, pueden ser escritas en términos de una multiplicación entre matrices [38], por lo que la expresión (2.16) sigue siendo válida para describir a una capa convolucional. Para lograr lo anterior, basta con asociar un *kernel* de convolución k_i a la capa i-ésima, y considerar una construcción especial para la matriz W_i a partir de los elementos de k_i [38, 39].

La motivación subyacente al uso de la operación de convolución en una red neuronal se basa, principalmente, en que explota la información topológica de las entradas para la extracción de características, por ejemplo, la correlación espacial presente en conjuntos de píxeles vecinos en una imágen [40]. El uso de la operación de convolución permite además una drástica reducción en el número de parámetros entrenables de una capa: corresponden tan solo a los elementos que constituyen al *kernel* k_i, el cual típicamente es más pequeño que la entrada sobre la cual la convolución es aplicada.

Las capas convolucionales a menudo emplean una operación no lineal adicional, denominada "*pooling*", la cual es aplicada sobre la salida de la función de activación de la capa. Esta operación entrega un estadístico sobre componentes de la salida que se encuentren dentro de en un cierto vecindario. A modo de ejemplo, la operación de *max pooling* entrega el valor máximo de los componentes vecinos, mientras que *average pooling* entrega su valor promedio.

[1] Las capas descritas anteriormente también se denominan *fully-connected*, pues cada una de sus unidades está "conectada" con todas las unidades de la capa anterior.

Agregar *pooling* a una capa convolucional permite el aprendizaje de características aproximadamente invariantes a pequeñas traslaciones de la entrada [36]. Alternativamente, es posible prescindir del uso de *pooling* empleando operaciones de convolución con *strides* incrementados [41], es decir, convoluciones donde el *kernel* se mueve en pasos mayores a uno, omitiendo algunas operaciones.

Redes neuronales recurrentes

Las redes neuronales recurrentes, a diferencia de las redes neuronales *feed-forward*, poseen al menos una conexión de retroalimentación [36], lo que causa que en ellas la información no fluya de forma unidireccional. Estas redes están especialmente diseñadas para procesar secuencias de datos, por ejemplo, series de tiempo.

Existen muchas formas en las que una red neuronal recurrente puede ser diseñada. En este trabajo nos centraremos en redes recurrentes que presentan conexiones de retroalimentación entre las salidas de sus capas ocultas, y que procesan secuencias de entradas $x^{(t)} \in \mathbb{R}^{l_x}$, con $t \in \{1, ..., T\}$. Para una red recurrente con estas características, una única capa oculta, y una inicialización $h^{(0)} \in \mathbb{R}^{l_h}$ para su estado oculto inicial, la salida $y^{(t)} \in \mathbb{R}^{l_y}$ asociada a la entrada $x^{(t)}$ puede ser calculada de acuerdo a (2.18) y (2.19), donde $W_{hx} \in \mathbb{R}^{l_h \times l_x}$, $W_{hh} \in \mathbb{R}^{l_h \times l_h}$, $W_{yh} \in \mathbb{R}^{l_y \times l_h}$, $b_h \in \mathbb{R}^{l_h}$ y $b_y \in \mathbb{R}^{l_y}$ definen transformaciones lineales, mientras que σ^h y σ^y corresponden a funciones de activación, típicamente no lineales, aplicadas elemento a elemento.

$$h^{(t)} = f^h(x^{(t)}, h^{(t-1)}) = \sigma^h(W_{hx}x^{(t)} + W_{hh}h^{(t-1)} + b_h) \tag{2.18}$$

$$y^{(t)} = f^y(h^{(t)}) = \sigma^y(W_{yh}h^{(t)} + b_y) \tag{2.19}$$

Los parámetros de esta red recurrente están dados por los elementos que componen a W_{hx}, W_{hh}, W_{yh}, b_h y b_y, y en el caso de utilizar funciones de activación paramétricas, por los parámetros de σ^h y/o σ^y, según corresponda. De forma similar al caso de las redes neuronales *feed-foward*, estos parámetros pueden estimados mediante la utilización conjunta de *Back Propagation Through Time* (BPTT), y SGD o alguna de sus variantes.

A partir de las expresiones (2.18) y (2.19), es posible descomponer los transformaciones realizadas por la red descrita en tres categorías: (i) sobre la entrada $x^{(t)}$ para el cómputo de $h^{(t)}$, (ii) sobre $h^{(t-1)}$ para el cómputo de $h^{(t)}$, y (iii) sobre $h^{(t)}$ para el cómputo de $y^{(t)}$ [36]. Cada una de estas categorías, no obstante, puede ser complejizada mediante la introducción de capas ocultas intermedias [36, 42].

El entrenamiento de las redes recurrentes descritas hasta ahora es desafiante, pues el aprendizaje de dependencias de largo plazo en secuencias de datos está asociado a problemas de desvanecimiento de gradientes y/o gradientes explosivos [43, 44]. Una forma de solucionar parcialmente este problema, consiste en la incorporación de conexiones de retroalimentación adicionales que favorezcan la propagación de información entre variables separadas por $c > 1$ pasos de tiempo [36]. La red recurrente relevante para el presente trabajo, "*Long Short-Term Memory*" (LSTM) [45, 46], implementa esta idea de forma efectiva, permitiendo eludir los problemas en el cálculo de gradientes que plagan a los modelos anteriormente descritos.

Las redes LSTM poseen mecanismos diseñados específicamente para abordar los problemas

asociados al aprendizaje de dependencias de largo plazo. Estas redes cuentan con una "memoria interna" $c^{(t)}$, cuya modificación es controlada mediante el uso de "compuertas". En una red LSTM moderna existen tres compuertas: de entrada ("*input gate*"), de olvido ("*forget gate*"), y de salida ("*output gate*"). Cada compuerta se comporta de forma análoga a la capa oculta definida en (2.18), tomando como entradas a un elemento $x^{(t)} \in \mathbb{R}^{l_x}$ de la secuencia a procesar, y al estado oculto anterior de la red LSTM, $h^{(t-1)} \in \mathbb{R}^{l_h}$. Formalmente, para $t \in \{1, ..., T\}$ y una cierta inicialización $h^{(0)} \in \mathbb{R}^{l_h}$ del estado oculto inicial, las salidas de las compuertas de entrada $i^{(t)} \in \mathbb{R}^{l_h}$, de olvido $f^{(t)} \in \mathbb{R}^{l_h}$, y de salida $o^{(t)} \in \mathbb{R}^{l_h}$ se calculan de acuerdo a (2.20), (2.21) y (2.22), respectivamente, donde W_{ix}, W_{fx}, $W_{ox} \in \mathbb{R}^{l_h \times l_x}$, W_{ih}, W_{fh}, $W_{oh} \in \mathbb{R}^{l_h \times l_h}$, y b_i, b_f, y $b_o \in \mathbb{R}^{l_h}$ definen transformaciones lineales, mientras que σ^i, σ^f y σ^o corresponden a funciones de activación sigmoidales.

$$i^{(t)} = \sigma^i(W_{ix}x^{(t)} + W_{ih}h^{(t-1)} + b_i) \tag{2.20}$$

$$f^{(t)} = \sigma^f(W_{fx}x^{(t)} + W_{fh}h^{(t-1)} + b_f) \tag{2.21}$$

$$o^{(t)} = \sigma^o(W_{ox}x^{(t)} + W_{oh}h^{(t-1)} + b_o) \tag{2.22}$$

En las redes LSTM, la información proveniente de $x^{(t)}$ y $h^{(t-1)}$ es condensada en una "memoria interna candidata", $\tilde{c}^{(t)} \in \mathbb{R}^{l_h}$, la cual es definida de forma análoga a las compuertas descritas anteriormente, pero reemplazando a la función sigmoidal de activación por una tangente hiperbólica. Más precisamente, $\tilde{c}^{(t)} \in \mathbb{R}^{l_h}$ es definida de acuerdo a (2.23), donde $W_{cx} \in \mathbb{R}^{l_h \times l_x}$, $W_{ch} \in \mathbb{R}^{l_h \times l_h}$ y $b_c \in \mathbb{R}^{l_h}$ definen una transformación lineal.

La memoria interna $c^{(t)} \in \mathbb{R}^{l_h}$ de la red LSTM es actualizada considerando la información nueva, contenida en $\tilde{c}^{(t)}$, e información pasada, contenida en su estado anterior $c^{(t-1)} \in \mathbb{R}^{l_h}$. Ambas fuentes de información poseen una importancia variable en la actualización de $c^{(t)}$, controlada por las compuertas de entrada $i^{(t)}$ y de olvido $f^{(t)}$, respectivamente. Formalmente, la regla de actualización de $c^{(t)}$ está dada por (2.24), donde $\odot$ denota la operación de multiplicación elemento a elemento. Finalmente, el nuevo estado interno de la red LSTM, $h^{(t)} \in \mathbb{R}^{l_h}$, es calculado según (2.25), esto es, aplicando la función tangente hiperbólica a la memoria interna actualizada, y multiplicando este resultado elemento a elemento con la compuerta de salida $o^{(t)}$.

$$\tilde{c}^{(t)} = \tanh(W_{cx}x^{(t)} + W_{ch}h^{(t-1)} + b_c) \tag{2.23}$$

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \tilde{c}^{(t)} \tag{2.24}$$

$$h^{(t)} = o^{(t)} \odot \tanh(c^{(t)}) \tag{2.25}$$

Red neuronal PointNet

PointNet es una red neuronal artificial especialmente diseñada para realizar clasificación o segmentación semántica de nubes de puntos [35]. Si bien en general una nube de puntos puede ser descrita como un conjunto no ordenado de vectores, en el caso de PointNet, el foco es puesto en nubes de puntos 3D, las que son representadas como conjuntos de la forma $\{p_j\}_{j=1}^{m}$, donde cada elemento p_j cuenta con ciertas coordenadas $(x_j, y_j, z_j) \in \mathbb{R}^3$ (aunque potencialmente podrían contar con información adicional, por ejemplo, un color).

Considerando que el orden de los elementos que componen a una nube de puntos no es relevante, PointNet ha sido diseñada para ser invariante a permutaciones sobre los elementos que componen

a las entradas que procesa. Para lograr lo anterior, la red busca aproximar una función general $f : 2^{\mathbb{R}^n} \to \mathbb{R}$ según (2.26), donde $h : \mathbb{R}^n \to \mathbb{R}^k$ y $g : \underbrace{\mathbb{R}^k \times \cdots \times \mathbb{R}^k}_{m} \to \mathbb{R}$ es una función simétrica.

$$f(\{p_1, p_2, ..., p_m\}) \approx g(h(p_1), h(p_2), ..., h(p_m)) = \hat{h}\left(\underset{j=1,...,m}{\text{MAX}} \{h(p_j)\}\right) \tag{2.26}$$

En la práctica, h corresponde a una red neuronal *feed-forward* y g a la composición entre una cierta función $\hat{h}$ (por ejemplo, otra red neuronal), y la operación de *max pooling* (que, en este caso, entrega un vector cuyas componentes son los máximos obtenidos al comparar los vectores de entrada elemento a elemento). De este modo, h codifica la información asociada a cada elemento de la nube de puntos, la operación de *max pooling* condensa esta información en un único vector de características global y, finalmente, la función $\hat{h}$ emplea esta información condensada como entrada. Para la tarea de clasificación de nubes de puntos, por ejemplo, $\hat{h}$ podría ser otra red neuronal entrenada como un clasificador de las características globales que se obtienen tras la operación de *max pooling*.

El módulo antes descrito ha sido utilizado como uno de los componentes fundamentales en la construcción de arquitecturas posteriores a PointNet, como PointNet++ [47], y guarda similitudes con módulos empleados por redes como SO-Net [48] o ShellNet [49]. En particular, este módulo es también empleado en los casos de estudio descritos en los Capítulos 3 y 4 de esta tesis.

Si bien PointNet, en su versión original, presenta decisiones de diseño y módulos adicionales orientados a la segmentación semántica de nubes de puntos (como la agregación de información local y global, y técnicas para que las representaciones aprendidas por la red sean invariantes a ciertas transformaciones), estas no son especialmente relevantes para el presente trabajo (mas pueden ser revisadas en [35]).

2.3.2. Deep Q-Network (DQN)

Deep Q-Network (DQN) es un algoritmo que fue desarrollado por Mnih et al. [29, 8] en el primer trabajo que demostró los beneficios de combinar aprendizaje profundo y aprendizaje reforzado. Este algoritmo fue capaz de aprender diversas políticas directamente a partir de observaciones de alta dimensionalidad, alcanzando con ellas desempeños de nivel humano en una gran variedad de juegos de Atari 2600.

DQN utiliza una red neuronal $Q_\theta(s, a)$ para aproximar a la función de valor óptima $Q^*(s, a)$ (ver Sección 2.2.1), y la emplea para derivar la política del agente. Esta red es entrenada utilizando una variante de Q-Learning [30] que incorpora dos modificaciones esenciales para lograr estabilizar el proceso de aprendizaje.

En DQN (al igual que en Q-Learning), la aproximación de la función $Q^*(s, a)$ se basa en la utilización de la ecuación de Bellman definida en (2.12) como una regla de actualización iterativa. Una opción para lograr lo anterior, consiste en minimizar el error cuadrático medio que puede ser definido al forzar que $Q_\theta(s, a)$ cumpla la relación de recurrencia que establece su ecuación de Bellman. Formalmente, la red $Q_\theta(s, a)$ puede ser entrenada minimizando una secuencia de funciones de costo $L_i(\theta_i)$, definidas de acuerdo a (2.27), donde θ_i corresponde al vector de parámetros de

$Q_\theta(s,a)$ tras i actualizaciones, $k \geq 1$, y $p_\pi(s,a)$ es la distribución de probabilidades del par (s,a), obtenida mediante la marginalización de la distribución $p_\pi(\tau)$ definida en (2.1).

$$L_i(\theta_i) = \mathbb{E}_{(s,a)\sim p_\pi(s,a)}\left[\left(r(s,a) + \gamma \max_{a'\in\mathcal{A}} \mathbb{E}_{s'\sim p(s'|s,a)} Q_{\theta_{i-k}}(s',a') - Q_{\theta_i}(s,a)\right)^2\right] \tag{2.27}$$

Denotando por y a la suma $r(s,a) + \gamma \max_{a'\in\mathcal{A}} \mathbb{E}_{s'\sim p(s'|s,a)} Q_{\theta_{i-k}}(s',a')$, y derivando el costo $L_i(\theta_i)$ con respecto a θ_i, el gradiente definido en (2.28) es obtenido. Con esto, el costo $L_i(\theta_i)$ puede ser minimizado utilizando, por ejemplo, SGD o alguna de sus variantes. En Q-Learning, la actualización de los parámetros del aproximador se realiza en cada paso de tiempo, empleando las transiciones instantáneas que experimenta el agente, y calculando y utilizando $k = 1$. Este enfoque de entrenamiento "*on-line*" presenta problemas de estabilidad [50, 29, 8] que DQN soluciona empleando dos mecanismos: *Experience Replay* [51] y *Target Networks* [8].

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{(s,a)\sim p_\pi(s,a)}\Big[\big(y - Q_{\theta_i}(s,a)\big)\nabla_{\theta_i} Q_{\theta_i}(s,a)\Big] \tag{2.28}$$

El uso de *Experience Replay* en DQN, consiste en el almacenamiento de transiciones de la forma (s_t, a_t, r_t, s_{t+1}), generadas por la interacción agente-ambiente en un arreglo circular finito D. Este arreglo es utilizado para muestrear uniformemente *minibatches* de transiciones, los cuales son utilizados para actualizar los parámetros de $Q_\theta(s,a)$ usando el gradiente definido en (2.29). La ventaja de este enfoque, comparado con su versión *on-line*, es que permite la reutilización de transiciones, rompe la correlación entre ellas, y evita que los parámetros de $Q_\theta(s,a)$ oscilen o caigan en mínimos locales [8].

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\Big[\big(y - Q_{\theta_i}(s,a)\big)\nabla_{\theta_i} Q_{\theta_i}(s,a)\Big] \tag{2.29}$$

El segundo mecanismo empleado por DQN, *Target Networks*, consiste en almacenar y utilizar una copia $Q_{\bar{\theta}}(s,a)$ de $Q_\theta(s,a)$ para el cómputo de y en (2.29), siendo esta copia actualizada clonando los parámetros θ de $Q_\theta(s,a)$ cada $k > 1$ actualizaciones. La utilización de este mecanismo reduce la posibilidad de que los parámetros θ presenten oscilaciones o diverjan, al introducir un retraso en el efecto que las actualizaciones de $Q_\theta(s,a)$ tienen sobre el cómputo de y [8].

DQN, al igual que Q-Learning, es un algoritmo *off-policy*: para lograr explorar el espacio de estados emplea una política ε-*greedy*, la cual selecciona acciones aleatorias con probabilidad ε, y acciones ávaras $a = \arg\max_{a'\in\mathcal{A}} Q_\theta(s,a')$ con probabilidad $(1-\varepsilon)$. El resumen completo del algoritmo se muestra en Algoritmo 1.

Es importante notar que la selección de acciones en DQN requiere la resolución de un problema de optimización: $a_t = \arg\max_{a\in\mathcal{A}} Q_\theta(s_t,a)$. Más aún, este problema debe ser resuelto múltiples veces durante el proceso de entrenamiento para el cómputo de y. Si $Q_\theta(s,a)$ establece un mapeo entre el par (s,a) y la estimación escalar de su retorno esperado, entonces se requieren $|\mathcal{A}|$ evaluaciones de $Q_\theta(s,a)$ para poder escoger acciones. La estrategia empleada en DQN, no obstante, consiste en mapear a cada estado s, el valor esperado del retorno de cada acción posible. Con esto, solo es necesaria una evaluación de $Q_\theta(s,a)$ para la selección de acciones: dadas las $|\mathcal{A}|$ salidas de la red neuronal, se selecciona la acción asociada a aquella salida cuyo valor sea el mayor.

Algoritmo 1: Deep Q-Network

Inicializar $Q_\theta(s,a)$ con parámetros θ
Inicializar $Q_{\bar{\theta}}(s,a)$ con parámetros $\bar{\theta} \leftarrow \theta$
Inicializar *replay memory* D
for *episodio* $= 1, M$ **do**
  Obtener s_1
  for $t = 1, T$ **do**
    Con probabilidad ε, eligir a_t aleatoriamente, si no $a_t = \arg\max_{a \in \mathcal{A}} Q_\theta(s_t, a)$
    Ejecutar acción a_t, observar r_t y s_{t+1}
    Guardar transición (s_t, a_t, r_t, s_{t+1}) en D
    Muestrear un *minibatch* de N transiciones (s_j, a_j, r_j, s_{j+1}) de D
    Calcular $y_j = \begin{cases} r_j & \text{si } s_{j+1} \text{ es un estado terminal,} \\ r_j + \gamma \max_{a'} Q_{\bar{\theta}}(s_{j+1}, a') & \text{si no.} \end{cases}$
    Actualizar $Q_\theta(s,a)$ minimizando el costo $L(\theta) = \frac{1}{N}\sum_{j=1}^{N}(y_j - Q_\theta(s_j, a_j))^2$
    Cada C actualizaciones de $Q_\theta(s,a)$, $\bar{\theta} \leftarrow \theta$
  end
end

2.3.3. Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) es un algoritmo de tipo actor-critico para control continuo propuesto por Lillicrap et al. [34]. Este algoritmo se basa en *Deterministic Policy Gradients* (DPG) [52], e incorpora las ideas que se propusieron en DQN para soslayar los problemas de estabilidad de Q-Learning: el uso de *Experience Replay* y *Target Networks*.

Una de las principales limitaciones de DQN, es que su aplicación está restringida a problemas con espacios de acciones discretos. La política que sigue el agente en DQN depende de la búsqueda de una acción $a = \arg\max_{a \in \mathcal{A}} Q_\theta(s,a)$ en cada paso de tiempo, procedimiento que deja de ser trivial cuando el espacio de acciones es continuo. Para sobrellevar esta limitación, DDPG mantiene un aproximador $Q_\theta(s,a)$ para la función de valor, y un aproximador $\mu_\phi(s)$ para la política (determinista) asociada, simplificando el proceso de selección de acciones a la evaluación de $\mu_\phi(s)$.

En DDPG, al igual que en DQN, la actualización de los parámetros de $Q_\theta(s,a)$ se basa en la utilización de la ecuación de Bellman para $Q^*(s,a)$. Considerando una política determinista $\mu_\phi(s)$, la expresión que define la secuencia de funciones de costo $L_i(\theta_i)$ que minimiza DQN puede ser adaptada a DDPG, quedando esta definida por (2.30). En consecuencia, denotando esta vez a y por la suma $r(s,a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} Q_{\theta_{i-k}}(s', \mu_\phi(s'))$, los gradientes asociados a los nuevos costos $L_i(\theta_i)$ de (2.30) quedan definidos por (2.28).

$$L_i(\theta_i) = \mathbb{E}_{(s,a)\sim p_\pi(s,a)}\left[\left(r(s,a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} Q_{\theta_{i-k}}(s', \mu_\phi(s')) - Q_{\theta_i}(s,a)\right)^2\right] \tag{2.30}$$

La regla de actualización para $\mu_\phi(s)$, por otro lado, se basa en la maximización del retorno esperado con respecto a los parámetros ϕ. Formalmente, este objetivo es aproximado siguiendo la

dirección del gradiente definido en (2.31).

$$\begin{aligned}\nabla_\phi J(\phi) &\approx \mathbb{E}_{s\sim p_\pi(s)}[\nabla_\phi Q_\theta(s, \mu_\phi(s))] \\ &= \mathbb{E}_{s\sim p_\pi(s)}[\nabla_a Q_\theta(s,a)|_{a=\mu_\phi(s)} \nabla_\phi \mu_\phi(s)]\end{aligned} \tag{2.31}$$

Como se mencionó anteriormente, al igual que DQN, DDPG utiliza *Experience Replay* y *Target Networks*. Las experiencias del agente son guardadas en un arreglo circular finito D, el cual es utilizado para muestrear *minibatches* de experiencias con las cuales actualizar los parámetros de $Q_\theta(s,a)$ y $\mu_\phi(s)$, es decir, la utilización de *Experience Replay* en DDPG es análoga a la de DQN.

En el caso de *Target Networks*, no obstante, DDPG se diferencia de DQN puesto que, en lugar de clonar la copia del aproximador cada cierto numero de actualizaciones, dicha copia es constantemente actualizada mediante un promedio móvil exponencial entre sus parámetros y los del aproximador original. En DDPG, además de $Q_\theta(s,a)$ y $\mu_\phi(s)$ se mantienen las redes "*target*" $Q_{\bar\theta}(s,a)$ y $\mu_{\bar\phi}(s)$, las cuales son empleadas para el cálculo de y en (2.29). Si los parámetros del aproximador original son denotados por ψ, y los de la copia por $\bar\psi$, entonces la regla de actualización de los parámetros de la copia está dada por $\bar\psi \leftarrow \lambda\psi + (1-\lambda)\bar\psi$, donde $\lambda \in [0,1]$ es el factor de suavizado del promedio móvil exponencial.

Al igual que DQN, DDPG es un algoritmo *off-policy*. Para facilitar la exploración del agente se seleccionan acciones utilizando una política $\mu_\mathcal{N}(s) = \mu_\phi(s) + \mathcal{N}$, donde $\mathcal{N}$ es un proceso aleatorio ("ruido de exploración"). El resumen del algoritmo se muestra en Algoritmo 2.

Algoritmo 2: Deep Deterministic Policy Gradient

Inicializar $Q_\theta(s,a)$ y $\mu_\phi(s)$ con pesos θ y ϕ
Inicializar $Q_{\bar\theta}(s,a)$ y $\mu_{\bar\phi}(s)$ con pesos $\bar\theta \leftarrow \theta$ y $\bar\phi \leftarrow \phi$
Inicializar *replay memory* D
for $episodio = 1, M$ **do**
  Inicializar un proceso aleatorio $\mathcal{N}$ para exploración
  Obtener s_1
  for $t = 1, T$ **do**
    Eligir $a_t = \mu_\phi(s_t) + \mathcal{N}_t$
    Ejecutar acción a_t, observar r_t y s_{t+1}
    Guardar transición (s_t, a_t, r_t, s_{t+1}) en D
    Muestrear un *minibatch* de N transiciones (s_j, a_j, r_j, s_{j+1}) de D
    Calcular $y_j = \begin{cases} r_j & \text{si } s_{j+1} \text{ es un estado terminal,} \\ r_j + \gamma Q_{\bar\theta}(s_{j+1}, \mu_{\bar\phi}(s_{j+1})) & \text{si no.} \end{cases}$
    Actualizar $Q_\theta(s,a)$ minimizando el costo $L(\theta) = \frac{1}{N}\sum_{j=1}^{N}(y_j - Q_\theta(s_j,a_j))^2$
    Actualizar $\mu_\phi(s)$ maximizando $J(\phi) = \frac{1}{N}\sum_j^N Q_\theta(s_j, \mu_\phi(s_j))$
    Actualizar *Target Networks*: $\bar\theta \leftarrow \lambda\theta + (1-\lambda)\bar\theta$; $\bar\phi \leftarrow \lambda\phi + (1-\lambda)\bar\phi$
  end
end

Además de DDPG, diversos algoritmos de aprendizaje reforzado para control continuo han sido propuestos. Algoritmos de tipo actor-crítico similares a DDPG incluyen, por ejemplo, a *Soft Actor-Critic* (SAC) [53] y al algoritmo *Twin Delayed Deep Deterministic Policy Gradient* (TD3) [54].

Capítulo 3

Evasión de colisiones mediante aprendizaje reforzado multimodal

En este capítulo se aborda el problema de evasión de colisiones para robots móviles, particularmente, para robots de servicio con limitadas capacidades de percepción. En concreto, se presenta una propuesta "*end-to-end*" que permite la obtención de una política de evasión de colisiones empleando aprendizaje reforzado. Esta política es parametrizada mediante una red neuronal artificial multimodal, y es capaz de entregar comandos de velocidad continuos para un robot omnidireccional en función de imágenes de profundidad, mediciones provenientes de sensores LiDAR, y estimaciones de velocidad basadas en odometría.

Con el fin de soslayar las limitaciones perceptuales con las que comúnmente cuentan los robots de servicio, se propone un método que permite la integración de mediciones de rango en el tiempo, y su aprovechamiento mediante un extractor de características *ad-hoc*, integrado a la parametrización de la política. Adicionalmente, para aliviar el problema del "*reality-gap*" que surge debido a que el entrenamiento de la política es conducido en simulaciones, se propone un *pipeline* de procesamiento para reducir las diferencias entre las imágenes de profundidad simuladas y reales.

Como caso de estudio, se considera el problema de evasión de colisiones en espacios de interior para Pepper [55], un robot humanoide que presenta importantes restricciones perceptuales. A través de pruebas simuladas, se muestra que el enfoque propuesto permite la obtención de una política de evasión de colisiones efectiva. Adicionalmente, se muestra empíricamente como la política obtenida puede ser transferida directamente al mundo real, donde se aprecia su capacidad de generalización al ser desplegada en diferentes ambientes.

3.1. Motivación

La capacidad de evadir colisiones contra obstáculos estáticos y/o dinámicos es una habilidad básica e indispensable para la operación segura de practicamente cualquier robot móvil. La resolución de este problema es un requerimiento fundamental para múltiples aplicaciones en robótica, por ejemplo, navegación autónoma, exploración autónoma, y coordinación multi-agente. Debido a la

importancia de este problema, una gran variedad de métodos han sido desarrollados para abordarlo (e.g. [56, 57, 58, 59, 60, 61]).

A grandes rasgos, los métodos de evasión de colisiones para robots móviles pueden ser categorizados como "globales" o "locales". Los métodos globales se caracterizan por usar un modelo global del ambiente (por ejemplo, un mapa de obstáculos), utilizarlo para planificar una trayectoria libres de colisiones (mediante una amplia gama de métodos [62, 63]), y posteriormente seguir dicha trayectoria. En contraposición, los sistemas locales emplean un modelo local del ambiente (típicamente construido a partir de información sensorial) para controlar al robot.

Sin importar su categorización, los sistemas clásicos de evasión de colisiones suelen ser modulares, por lo que su funcionamiento está basado en la interacción de sub-sistemas que abordan diferentes partes del problema. En términos generales, estos sub-sistemas se encargan de la extracción y procesamiento de información del ambiente, y luego, a través de su explotación, de la planificación y/o ejecución de trayectorias libres de colisiones. La principal desventaja de este enfoque modular, es que cada sub-sistema, en general, posee un número de parámetros considerable, cuyo ajuste suele depender fuertemente del robot utilizado, y de las características del ambiente en el que dicho robot es desplegado.

Una alternativa al enfoque modular adoptado por los sistemas clásicos, corresponde al enfoque basado en aprendizaje. Este enfoque busca resolver el problema de evasión de colisiones sin descomponerlo en sub-problemas, mediante el aprendizaje de un mapeo entre la información que caracteriza al ambiente, y las acciones que un robot debería ejecutar para evadir colisiones dada esta información. Adicionalmente, este enfoque permite la obtención de sistemas reactivos, pues su comportamiento depende directamente de información instantánea del entorno. El éxito previo que métodos basados en aprendizaje han tenido en problemas diferentes, motiva intentar emplearlos para resolver el problema de evasión de colisiones para robótica móvil.

3.2. Trabajos relacionados

Como se señaló en la sección anterior, la evasión de colisiones en robótica es un problema que ha sido abordado por una gran variedad de métodos. Los métodos basados en aprendizaje, en particular, han cobrado gran relevancia en el último tiempo, posiblemente debido a la serie de avances que el aprendizaje profundo ha permitido en múltiples campos.

En este contexto, el problema de evasión de colisiones puede ser formulado como un problema de aprendizaje supervisado, donde se busca aprender un mapeo entre observaciones y acciones en función de una serie de datos etiquetados. Por ejemplo, en [64], una política de evasión de colisiones parametrizada mediante una red neuronal convolucional es entrenada de forma supervisada. Esta política logra mapear imágenes de profundidad, a ponderadores discretos para la generación de comandos de velocidad para un robot diferencial. Un enfoque similar es propuesto en [65], donde una red neuronal convolucional es entrenada con el fin de mapear imágenes de profundidad y de normales a una trayectoria (elegida a partir de un conjunto discreto de opciones) para controlar un *quadrotor*. En este caso, las imágenes de profundidad y de normales son inferidas a partir de imágenes RGB empleando otra red convolucional, también entrenada de manera supervisada.

Uno de los principales desafíos técnicos asociados al entrenamiento de esta clase de políticas, es que requieren una gran cantidad de datos etiquetados. Más aún, determinar si la base de datos es representativa, y sus etiquetas permiten encapsular correctamente el objetivo de aprendizaje (en este caso, evadir obstáculos), es también problemático.

Una alternativa al enfoque descrito, consiste en plantear el problema de evasión de colisiones dentro del marco de trabajo del aprendizaje reforzado. Empleando aprendizaje reforzado, la necesidad de contar con datos etiquetados es eliminada, por lo que el desempeño de la política entrenada deja de estar acotado por la habilidad de un supervisor; no obstante, usualmente un número significativo de interacciones agente-ambiente es necesario para lograr adquirir los comportamientos deseados, por lo cual es común que bajo este enfoque, las políticas sean entrenadas en simulación.

Una gran cantidad de trabajos han abordado el problema de evasión de colisiones en robótica utilizando aprendizaje reforzado (e.g. [66, 67, 68, 69, 22, 61, 70, 71, 72, 73]). Varios de estos trabajos formulan el problema de evasión de colisiones (o tareas relacionadas, como por ejemplo navegación) considerando que el agente cuenta con observaciones construídas a partir de mediciones provenientes de LiDARs 2D (e.g. [66, 67, 23]), o provenientes de cámaras (e.g. [67, 69, 70]). En términos generales, el diseño de las observaciones del agente depende de su sensorización. En algunos casos, depender de una única fuente de información puede ser problemático si es que esto limita fuertemente la observabilidad del ambiente (esto ocurre en el presente caso de estudio, usando al robot Pepper, como se verá en la Sección 3.3.1). Para mitigar este problema, en este trabajo se utiliza una red neuronal multimodal para la parametrización de la política [74, 75, 76]. Utilizando esta clase de redes neuronales, es posible fusionar la información de múltiples sensores, lo que permite generar observaciones más informativas del estado del ambiente y, bajos ciertas condiciones, hacer que estas observaciones sean robustas a perturbaciones sobre las mediciones de los sensores [74, 76]. En particular, la red neuronal multimodal diseñada en este trabajo procesa imágenes de profundidad, mediciones provenientes de LiDARs 2D, y estimaciones de velocidad basadas en odometría.

En ambientes relativamente simples, agentes con restricciones perceptuales importantes pueden demostrar comportamientos de evasión de colisiones razonables tras ser entrenados. No obstante, en general, estas restricciones perceptuales deben ser abordadas si el agente debe ser desplegado en un ambiente complejo. Un enfoque directo para lograr lo anterior, consiste en simplemente instalar sensores adicionales para mejorar las capacidades perceptuales del agente y, consecuentemente, rediseñar las observaciones que este recibe para que incorporen la información extra que estos sensores proveen (por ejemplo, empleando parametrizaciones multimodales como se señaló anteriormente). En ocasiones, no obstante, la instalación de sensores adicionales es inviable por restricciones físicas y/o monetarias. Una forma de aliviar la observabilidad parcial en estos casos, consiste en emplear una red neuronal recurrente para parametrizar a la política [77]. En este trabajo, esta decisión de diseño también es adoptada, con lo que la política entrenada es a la vez multimodal y recurrente. Para complementar estas estrategias, adicionalmente, un método simple de agregación de mediciones en el tiempo es propuesto. Este método permite mantener una representación de regiones fuera del rango de percepción instantáneo del agente, y aumentar artificialmente la resolución de sus observaciones a medida que este se desplaza.

Finalmente, en este caso de estudio también se aborda el problema asociado a la transferencia de las políticas entrenadas en ambientes simulados, a ambientes reales, tratando de aliviar las di-

ferencias entre las observaciones del agente en simulaciones, y las observaciones del agente en el mundo real. Por lo general, estas diferencias son importantes al considerar sensores cuyas mediciones no son fácilmente simulables a un nivel realista, tales como cámaras RGB. Este es uno de los motivos por el cual la mayoría de los trabajos previos en evasión de colisiones que optan por la utilización de imágenes como observaciones, utilizan sensores de profundidad, o transforman la imagen original a una representación intermedia más simple. Algunos métodos alternativos para reducir la brecha entre imágenes simuladas y reales, consisten en el uso de estrategias basadas en modelos generativos [78], no obstante, su utilización implica, en muchos casos, la necesidad de una base de datos etiquetados.

En este trabajo, las imágenes de profundidad empleadas como parte de las observaciones del agente son procesadas de acuerdo a un *pipeline* clásico, el cual es propuesto con el fin de abordar las importantes diferencias entre las imágenes de profundidad simuladas y reales que son utilizadas: mientras que las imágenes simuladas son ideales, las reales a menudo presentan distorsiones y pérdidas de información. En este contexto, métodos clásicos para la reconstrucción de imágenes de profundidad reales incluyen el estudio y caracterización de modelos de su ruido (e.g. [79, 22]). El método propuesto, no obstante, está más estrechamente relacionado al presentado en [80], del cual fueron tomadas varias de las ideas generales que encauzaron su desarrollo.

3.3. Evasión de colisiones para el robot Pepper

La evasión de colisiones para robots móviles puede ser modelada como un problema de toma de decisiones secuenciales, donde un agente (el robot) interactúa con el ambiente en pasos de tiempo discretos, ejecutando acciones que le permitan desplazarse por regiones navegables libres de obstáculos. En este trabajo, la interacción agente-ambiente es modelada como un Proceso de Decisión de Markov parcialmente observable (POMDP, ver Sección 2.1.1), el agente es el robot Pepper [55], y los ambientes considerados corresponden a ambientes de interior.

Pepper es un robot humanoide especialmente diseñado para interactuar con humanos. Actualmente es empleado como robot asistente y/o recepcionista en diferentes rubros comerciales. Adicionalmente, Pepper es utilizado en la liga de plataforma social estándar de la competencia internacional RoboCup@Home desde el 2017, año en que coincidentemente dicha liga fue introducida [81].

La Figura 3.1 muestra los sensores del robot Pepper v1.6[1] que resultan ser relevantes para este caso de estudio. Como se ilustra, este robot posee dos cámaras RGB y un sensor de profundidad en la cabeza. Además, cuenta con tres sensores laser situados cerca de su base omnidireccional, y con tres actuadores laser que permiten la caracterización de su entorno. El listado completo de los sensores y actuadores de Pepper no es relevante para este trabajo, no obstante, puede ser encontrado en [55].

Las características de Pepper hacen que su uso no sea apropiado para cualquier aplicación. Los sensores y actuadores con los que ha sido equipado han sido pensados para casos de uso específicos, centrados en interacción humano-robot [55]. Por ejemplo, a pesar de contar con brazos, estos han

[1] Actualmente existen tres versions de Pepper: v1.6, v1.8 y v1.8a.

sido diseñados para dotar al robot de expresividad, siendo sus capacidades de manipulación muy limitadas [55]. Como consecuencia, dotar a Pepper con la capacidad de navegar resulta ser una tarea muy desafiante: el robot posee fuertes restricciones perceptuales por la disposición y características de sus sensores exteroceptivos.

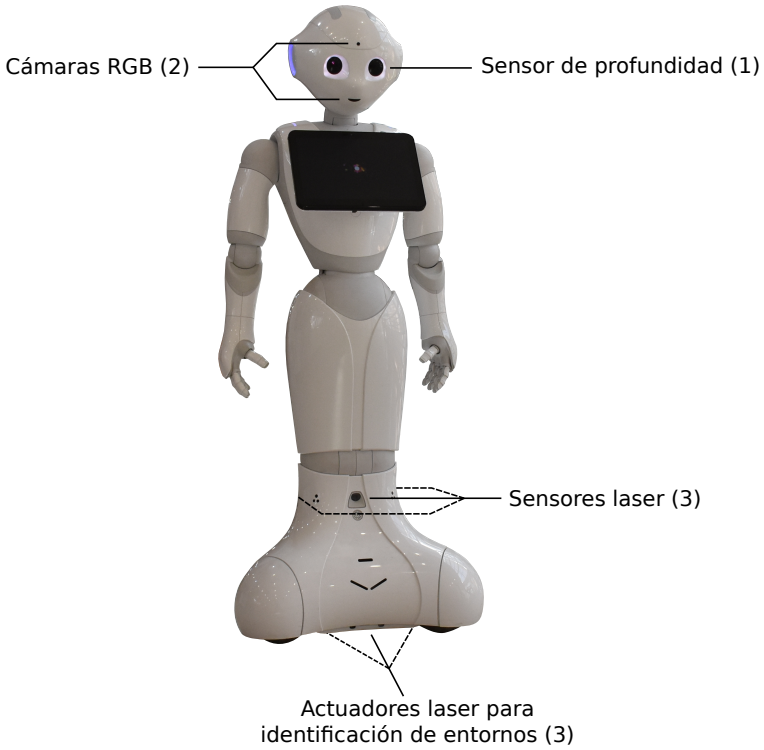


Figura 3.1: Robot humanoide Pepper v1.6, y sus sensores exteroceptivos principales. Las líneas punteadas indican que el sensor señalado no es directamente visible en la imagen, por encontrarse en el dorso o la parte trasera del robot. Para cada etiqueta, el número en paréntesis indica la cantidad de sensores o actuadores, según corresponda.

3.3.1. Modelamiento del problema

Para resolver el problema empleando aprendizaje reforzado, la interacción agente-ambiente es modelada como un POMDP definido por la tupla $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \Omega, \mathcal{O})$. Bajo esta formulación, los estados $s_t \in \mathcal{S}$, y las funciones de observación $\mathcal{O}$ y de transición de estados $\mathcal{T}$ quedan determinados por el ambiente y su dinámica. Las observaciones $o_t \in \Omega$ y acciones $a_t \in \mathcal{A}$, no obstante, pueden ser diseñadas de manera más flexible considerando las restricciones de percepción y actuación del agente. Por último, la función de recompensa r puede ser diseñada de forma arbitraria para guiar el aprendizaje de la política.

Observaciones

Tomando en consideración los sensores con los que cuenta el robot Pepper [55], las observaciones son construidas a partir del procesamiento de imágenes de profundidad, mediciones de rango

provenientes de sensores LiDAR, y estimaciones de velocidad basadas en odometría. Si bien Pepper cuenta con dos cámaras RGB, la utilización de las imágenes que estos sensores proveen es descartada. Lo anterior, debido a las sustanciales diferencias entre imágenes RGB reales y simuladas, y a la gran variabilidad de entornos en los que se espera que la política entrenada sea desplegable.

Con lo anterior, una observación en el instante de tiempo t es definida como la tupla $o_t = (o^t_{\text{depth}}, o^t_{\text{range}}, o^t_{\text{odom}})$, donde o^t_{depth} corresponde a la imagen de profundidad pre-procesada capturada por el robot, o^t_{range} corresponde a mediciones pre-procesadas provenientes de los sensores LiDAR localizados cerca de su base, y o^t_{odom} a estimaciones de su velocidad, basadas en odometría. Mientras que las imágenes de profundidad son procesadas para reducir las diferencias entre aquellas que son simuladas y aquellas que son obtenidas en el mundo real, y de este modo abordar el problema del "*reality-gap*" (ver Sección 3.3.2), las mediciones provenientes de los sensores LiDAR son procesadas para mejorar artificialmente las capacidades perceptuales de Pepper (ver Sección 3.3.3).

Acciones

Al igual que en varios trabajos sobre evasión de colisiones empleando aprendizaje de máquinas, en este trabajo, las acciones corresponden a comandos de velocidad continuos para un controlador de bajo nivel. Puesto que Pepper es un robot omnidireccional, una acción dada para el instante de tiempo t es definida como $a_t = (v^t_x, v^t_y, v^t_\theta)$, donde $v^t_x \in [v^{\text{min}}_x, v^{\text{max}}_x]$ corresponde a la velocidad lineal del robot en el eje x, $v^t_y \in [v^{\text{min}}_y, v^{\text{max}}_y]$ a su velocidad lineal en el eje y, y $v^t_\theta \in [v^{\text{min}}_\theta, v^{\text{max}}_\theta]$ a su velocidad angular (en torno al eje z). Adicionalmente, los límites para las componentes de a_t son fijados de manera tal que $v^{\text{max}}_\alpha > 0$ y $v^{\text{min}}_\alpha = -v^{\text{max}}_\alpha$, para $\alpha \in \{x, y, \theta\}$ (los valores numéricos fijados para estos límites son reportados en la Tabla 3.1).

Recompensa

La función de recompensa es diseñada con el fin de promover el movimiento del robot en el ambiente, penalizando maniobras arriesgadas y colisiones. Se define de acuerdo a (3.1), donde l^t_{min} corresponde al menor rango medido con los sensores LiDAR cercanos a la base del robot en el tiempo t, y l_{thresh} es una distancia de umbral fija.

$$r_t = \begin{cases} r^t_{\text{collision}} & \text{si el robot colisiona} \\ r^t_{\text{danger}} & \text{si } l^t_{\text{min}} < l_{\text{thresh}} \\ r^t_{\text{move}} & \text{si no} \end{cases} \tag{3.1}$$

El término $r^t_{\text{collision}}$ busca penalizar fuertemente las colisiones del agente, por lo que toma el valor de -10. El término r^t_{danger}, por otro lado, cumple el rol de penalizar al agente cuando este se acerca peligrosamente a obstáculos. Tomando en consideración la geometría de Pepper, el umbral de distancia l_{thresh} es fijado en 0.6 metros. Con lo anterior, r^t_{danger} es definido según (3.2), donde $K = 1.5$ es un factor escalar fijo.

$$r^t_{\text{danger}} = K(l^t_{\text{min}} - 1) \tag{3.2}$$

Finalmente, el término r_{move} es definido según (3.3) y (3.4). Este término busca incentivar al agente a movilizarse por el ambiente, penalizando velocidades y aceleraciones angulares altas, y también velocidades negativas en el eje x. Mientras que las velocidades y aceleraciones angulares altas son penalizadas para evitar comportamientos indeseables (como que el agente gire en un sitio fijo), las velocidades negativas en el eje x son penalizadas tomando en cuenta que los sensores exteroceptivos empleados para la construcción de las observaciones del agente (sensor de profundidad y sensores laser ubicados cerca de su base) no entregan la información necesaria como para caracterizar apropiadamente obstáculos que estén posicionados detrás del robot. Adicionalmente, la penalización sobre velocidades negativas en el eje x es justificada considerando que un desplazamiento en reversa resultaría poco natural para un robot social como Pepper.

$$r^t_{\text{move}} = \begin{cases} \dfrac{v^t_x}{v^{\max}_x} \cos\left(\dfrac{\pi}{2}\dfrac{K^t_{v_\theta}}{v^{\max}_\theta}\right) & \text{si } v^t_x \geq 0 \\ \dfrac{v^t_x}{v^{\max}_x} & \text{si } v^t_x < 0 \end{cases} \tag{3.3}$$

$$K^t_{v_\theta} = \text{mín}\{\text{máx}\{|v^t_\theta|, |v^t_\theta - v^{t-1}_\theta|\}, v^{\max}_\theta\} \tag{3.4}$$

Para comprender la definición de r^t_{move} primero hay que notar que $K^t_{v_\theta}$ es el término encargado de penalizar velocidades y aceleraciones angulares. De (3.4), y considerando que $v^{\min}_\theta = -v^{\max}_\theta$, es claro que $K^t_{v_\theta} \in [0, v^{\max}_\theta]$, siendo este término nulo solo cuando $|v^t_\theta| = 0$ y $|v^t_\theta - v^{t-1}_\theta| = 0$, y alcanzando su máximo valor cuando $|v^t_\theta| = v^{\max}_\theta$, o en el caso en que $|v^t_\theta - v^{t-1}_\theta| \geq v^{\max}_\theta$. En consecuencia, el término $\cos(\pi K^t_{v_\theta}/(2v^{\max}_\theta))$ es nulo cuando $K^t_{v_\theta}$ alcanza su valor máximo, e igual a uno cuando $K^t_{v_\theta}$ es igual a cero.

Con lo anterior, y notando que $v^t_x/v^{\max}_x \in [-1, 1]$, de (3.3) se desprende que si $v^t_x \geq 0$, entonces $r^t_{\text{move}} \in [0, 1]$ y el término $\cos(\pi K^t_{v_\theta}/(2v^{\max}_\theta))$ actúa modulando la recompensa obtenida en función de las velocidades y aceleraciones angulares del agente. En contraposición, si $v^t_x < 0$, entonces $r^t_{\text{move}} \in [-1, 0)$ y las velocidades y aceleraciones angulares no tienen efecto en su cómputo.

Finalmente, si bien Pepper es un robot omnidireccional, hay que notar que la velocidad lineal del agente en el eje y, v^t_y, no es incluida en el cómputo de r_t. Esta decisión de diseño busca que la política entrenada controle esta componente de la velocidad del robot de forma libre.

3.3.2. Procesamiento de imágenes de profundidad

Existen diferencias sustanciales entre las imágenes de profundidad simuladas para el entrenamiento de la política, y aquellas entregadas por el sensor de profundidad de Pepper en el mundo real. Las imágenes de profundidad simuladas en Gazebo se caracterizan por ser ideales: no poseen ruido, distorsiones, ni pérdidas de información. En la realidad, no obstante, el sensor de profundidad de Pepper entrega imágenes con múltiples aglomeraciones de píxeles con valores desconocidos, y con regiones donde la estimación de la distancia es errada, o presenta distorsiones. Debido a estas diferencias, una política entrenada en simulaciones presentaría una notable baja en rendimiento al ser desplegada en el mundo real. Con el fin de aliviar este problema, tanto las imágenes de profundidad simuladas como las reales son sometidas a una etapa de pre-procesamiento.

El sensor de profundidad de Pepper funciona empleando luz estructurada en el espectro infrarrojo cercano: mediante la emisión de un patrón de luz y el análisis de su deformación al impactar sobre diferentes superficies, se realiza la estimación de un mapa de profundidad de la escena. Debido al uso de luz estructurada, este sensor solo genera mapas de profundidad adecuados en ambientes de interior o en ambientes con iluminación controlada. Incluso si no existen fuentes externas de luz que interfieran con la lectura del patrón emitido por el sensor, la reflectividad de algunos objetos inevitablemente hará que las estimaciones de profundidad asociadas a ciertas regiones de la escena sean ruidosas o infactibles. Las pérdidas de información inherentes al principio de funcionamiento de este sensor, hacen que la reconstrucción de imágenes de profundidad ideales, semejantes a aquellas generadas en simulación, sea una tarea muy desafiante.

Aceptando que las mejoras aplicables sobre las imágenes de profundidad reales son limitadas, sus diferencias con las imágenes simuladas podrían disminuirse al introducir distorsiones sobre estas últimas. En este trabajo se opta por seguir este enfoque, sin descartar completamente el mejoramiento de las imágenes reales: primero se introducen distorsiones sobre las imágenes simuladas para que se asemejan a las reales, y posteriormente, tanto las imágenes reales como las simuladas son procesadas para aliviar estas distorsiones.

La introducción de distorsiones en las imágenes de profundidad simuladas requiere una caracterización de las distorsiones presentes en las imágenes obtenidas por el sensor real. Si bien el sensor de profundidad de Pepper entrega imágenes imperfectas debido a múltiples factores, el efecto de estos factores (a grandes rasgos) se refleja en píxeles con valores incorrectos y en píxeles con valores desconocidos. Estos últimos a menudo forman parches irregulares que cubren un porcentaje importante de las imágenes obtenidas, por lo que abordar su existencia resulta esencial para reducir las diferencias entre las imágenes de profundidad reales y simuladas.

Debido a que las estimaciones de profundidad realizadas por el sensor real se basan en el procesamiento de patrones de luz estructurada, es usual que píxeles con valores incorrectos y/o desconocidos aparezcan en los contornos de diferentes objetos presentes en una escena. Con el fin de encontrar estas regiones en las imágenes simuladas, a estas primeramente se les aplica un detector de bordes de Canny [82].

Desafortunadamente, la extracción de todos los contornos presentes en un mapa de profundidad simulado no siempre es factible utilizando solo el detector de bordes de Canny. Para abordar este problema, y aprovechando la flexibilidad que ofrece el entorno de simulación utilizado, este detector también es aplicado sobre las imágenes RGB correspondientes a las imágenes de profundidad simuladas. Los bordes detectados en ambos tipos de imágenes son posteriormente agregados, con lo que se obtiene una representación más completa de los bordes presentes en la escena.

Algunos de los bordes detectados en las imágenes RGB simuladas, no obstante, podrían surgir meramente debido a la presencia de texturas, en cuyo caso deberían ser descartados. Para poder filtrar estos bordes, primeramente se buscan regularidades en el mapa de contornos agregados, aplicando una transformada de Hough [83] sobre este. La transformada aplicada se utiliza para encontrar líneas rectas en los contornos detectados, y posteriormente realizar una comparación entre los valores de los píxeles de la imagen de profundidad que se encuentren alineados en paralelo a las rectas detectadas. Si el valor absoluto de la diferencia entre estos píxeles no supera un cierto umbral, entonces el borde asociado probablemente fue detectado debido a la presencia de texturas en la simulación, y consecuentemente, es descartado.

El mapa de contornos resultante resalta las delimitaciones geométricas de los objetos presentes en la escena observada. Para introducir corrupción en las imágenes de profundidad simuladas, al mapa de contornos primero se le aplica la operación morfológica de dilación, y a partir de parches de píxeles de tamaño aleatorio, marcados en posiciones que concuerden con el mapa de contornos dilatado, una máscara binaria es construida. Está máscara representa la distribución espacial de píxeles corruptos que podría encontrarse en una imagen de profundidad real, por lo que es aplicada sobre la imagen de profundidad simulada asociada.

Para abordar la pérdida de información presente en las imágenes de profundidad reales y en las simuladas pre-procesadas, el valor de los píxeles desconocidos es predicho empleando el algoritmo de restauración de Telea [84]. Finalmente, tanto las imágenes de profundidad reales y simuladas resultantes, son filtradas (para evitar *aliasing*) y submuestreadas para reducir sus dimensiones desde 320×240, a 80×60 píxeles. La Figura 3.2 muestra un diagrama del procedimiento descrito.

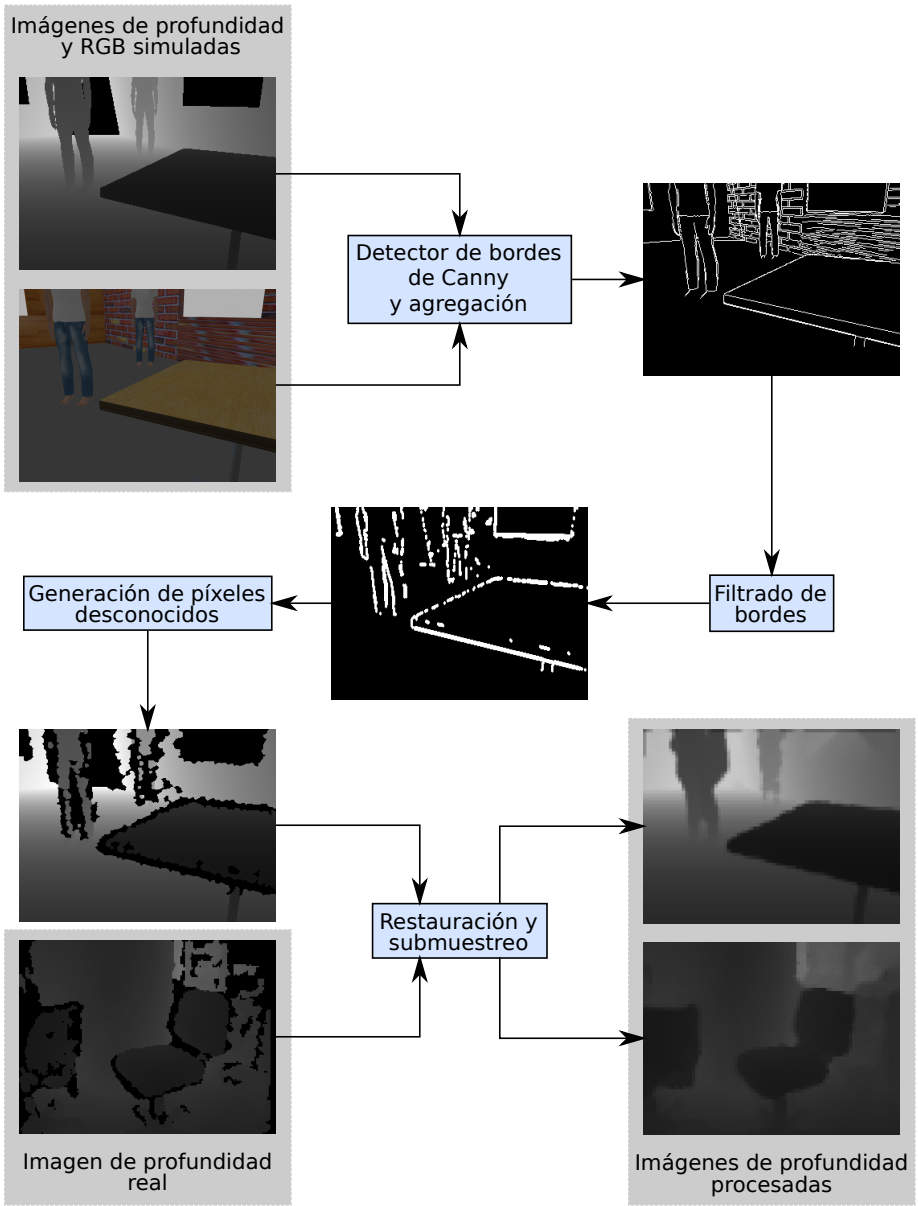


Figura 3.2: Diagrama del procesamiento aplicado sobre las imágenes simuladas y reales. Adaptado de [25].

3.3.3. Procesamiento de mediciones de rango

El arreglo de sensores LiDAR situado cerca de la base omnidireccional de Pepper provee una caracterización muy limitada del ambiente. Este arreglo se compone de tres sensores, cada uno de ellos entregando 15 mediciones dentro de un campo de visión (*Field of View* [FoV]) horizontal de 60°. Mientras que un sensor está alineado al eje x del robot, los otros dos se separan de este por una cierta distancia angular (ver Figura 3.3). Naturalmente, la disposición geométrica de estos sensores genera regiones ciegas en el campo de percepción del robot.

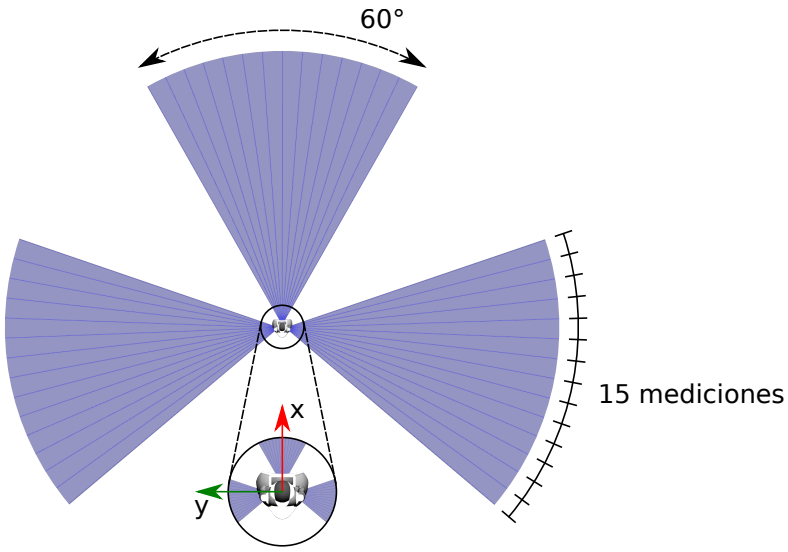


Figura 3.3: Diagrama del arreglo de sensores LiDAR del robot Pepper. Vista superior.

Para abordar este problema, las mediciones instantáneas obtenidas a partir de los sensores LiDAR son agregadas en un "mapa local" del ambiente. Este mapa local es construido integrando las lecturas de rango de los LiDAR en el tiempo, usando una referencia fija, y representando las mediciones agregadas como una nube de puntos ("*point cloud*"). El marco de referencia empleado para la integración de las mediciones, corresponde al entregado por el sistema de odometría de Pepper.

Las estimaciones de posición obtenidas mediante odometría, integran los errores de medición de los codificadores rotatorios asociados a la base omnidireccional del robot. En consecuencia, existe el riesgo de producir mapas locales distorsionados si la región considerada para su construcción es demasiado extensa. Por lo anterior, la integración de mediciones se restringe a un radio D, medido desde la posición del robot para un instante de tiempo dado. En el caso de Pepper, este radio es fijado en 6.5 metros.

Naturalmente, la integración de mediciones en el tiempo que fue descrita no es suficiente para obtener una buena representación del estado del ambiente, pues asume que este es completamente estático. Para abordar esta limitación, un proceso simple de filtrado es aplicado sobre las mediciones integradas, el cual descarta aquellas que no son consistentes con las lecturas instantáneas de los sensores. Este proceso también filtra información redundante considerando un umbral de tolerancia angular. Un ejemplo del mapa local descrito es mostrado en la Figura 3.4, donde, dada una cierta trayectoria seguida por el robot, se muestra como esta representación se compara con la que sería obtenida solo empleando mediciones de rango instantáneas.

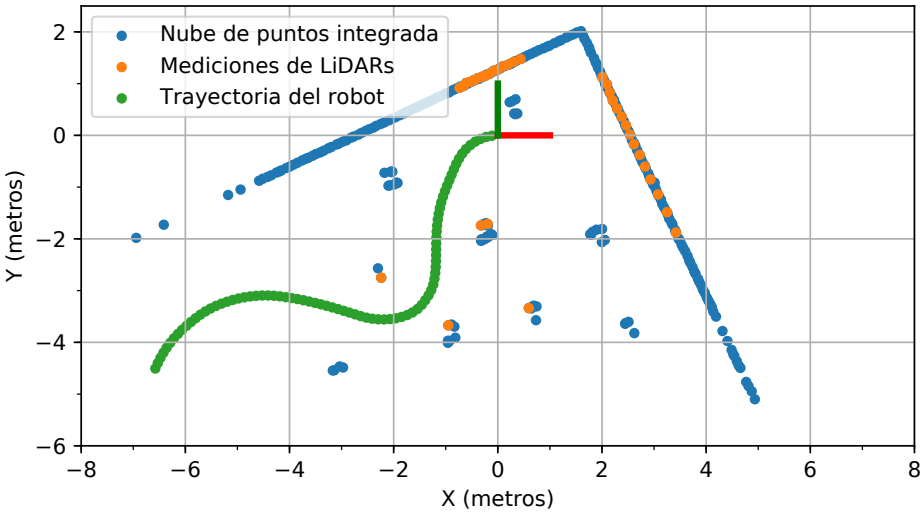


Figura 3.4: Comparación entre el "mapa local" propuesto, y mediciones instantáneas de rango, considerando una trayectoria dada. Los puntos verdes representan la trayectoria seguida por el robot, mientras que los ejes con origen en $(0, 0)$ definen su pose en el tiempo actual (en rojo, eje x, en verde, eje y). Los puntos naranja representan las mediciones instantáneas obtenidas por los sensores LiDAR cercanos a la base omnidireccional de Pepper. Los puntos azules corresponden a la representación propuesta, la cual es construída integrando y filtrando las mediciones instantáneas obtenidas a lo largo de la trayectoria seguida por el robot. Adaptado de [25].

El uso del mapa local propuesto permite aliviar las limitaciones perceptuales de Pepper, modelando el ambiente con una precisión que incrementa en el tiempo, a medida que el robot se desplaza. Adicionalmente, y a diferencia de una representación vectorial simple de mediciones de rango, no tiene limitaciones en relación a su dimensión (la cual puede variar en cada instante de tiempo), ni requiere una codificación explícita para mediciones fuera de rango. Un análisis detallado de las ventajas de esta representación es desarrollada en el Capítulo 4.

3.3.4. Algoritmo y parametrización de la política

Para entrenar la política de evasión de colisiones, el algoritmo DDPG [34] es utilizado (ver Sección 2.3.3). Al ser DDPG un algoritmo de tipo actor-crítico, dos redes neuronales independientes son empleadas para parametrizar a la política (actor) y a la función de valor del par estado-acción (crítico), respectivamente.

Las observaciones recibidas por el agente son construidas a partir de fuentes heterogéneas de información, y representadas a través de estructuras de datos diferentes (ver Sección 3.3.1). En efecto, estas se definen como $o_t = (o^t_{\text{depth}}, o^t_{\text{range}}, o^t_{\text{odom}})$, donde o^t_{depth} es una imagen de profundidad pre-procesada (ver Sección 3.3.2), o^t_{range} es una nube de puntos (ver Sección 3.3.3), y o^t_{odom} es un vector con estimaciones de las componentes instantáneas de velocidad del robot, obtenidas mediante odometría. Para procesar la información contenida en estas observaciones, dadas sus características, se adopta un enfoque multimodal: cada componente de o_t es procesado de forma independiente por extractores de características *ad-hoc*, y las representaciones intermedias resultantes son concatenadas.

Al ser o^t_{depth} una imagen, su extractor de características consiste simplemente en una secuencia de capas convolucionales. De manera similar, al ser o^t_{odom} un vector, su extractor de características consiste en una secuencia de capas *fully connected*.

Mientras que o^t_{depth} y o^t_{odom} poseen dimensiones fijas, al ser o^t_{range} una nube de puntos de cardinalidad variable, las capas convolucionales o *fully connected* no son opciones viables para procesarla de forma directa. Para hacerlo, en cambio, se sigue el enfoque propuesto en PointNet [35], es decir, se utilizan capas *fully connected* para el procesamiento de cada elemento perteneciente a o^t_{range}, y una función de activación simétrica condensa la información obtenida en una representación de dimensión fija. Más precisamente, cada punto de o^t_{range} es procesado por la misma red *feed-forward*, y las salidas resultantes son agregadas aplicando la operación de *max pooling*.

Las arquitecturas de las parametrizaciones para el actor y el crítico se resumen en la Figura 3.5. A pesar de que las redes consideradas en este trabajo son multimodales, su estructura general y algunos de los parámetros de sus capas se basan en aquellos empleados en [77]. Como puede observarse en la Figura 3.5, las salidas de los extractores de características que procesan a cada componente de las observaciones son concatenadas, y procesadas por una nueva secuencia de capas, dentro de las cuales se incluye una capa LSTM. La salida de la última capa corresponde a la acción que deberá ejecutar el agente, en el caso del actor, y al valor esperado del retorno del par estado-acción, en el caso del crítico. El uso de la función tanh para la capa de salida del actor acota los valores numéricos de las componentes de las acciones al intervalo $[-1, 1]$, lo que permite luego escalarlos trivialmente al intervalo determinado por $v^{\min}_\alpha$ y $v^{\max}_\alpha$, para $\alpha \in \{x, y, \theta\}$. En el caso del crítico, la función de activación asociada a la capa de salida es lineal considerando que esta red busca aproximar valores reales en un cierto intervalo numérico[2].

Las parametrizaciones para el actor y el crítico incluyen capas LSTM con el fin de aliviar la observabilidad parcial causada, en parte, por las limitaciones perceptuales de Pepper. De este modo, mediante la integración de información proveniente de observaciones recibidas por el agente en cada paso de tiempo, se busca aproximar el estado del ambiente. Al ser las redes del actor y el crítico recurrentes, no es adecuado realizar muestreos uniformes de experiencias del *Experience Replay* para actualizar sus pesos. En cambio, se sigue la metodología empleada en [77], y validada exhaustivamente en [85]: las experiencias son muestreadas en secuencias de tamaño fijo, y las actualizaciones sobre los pesos de las redes omiten los primeros elementos de las secuencias, con el fin de inicializar un estado interno para las capas LSTM.

3.4. Evaluación experimental

El entrenamiento de las políticas es realizado en simulaciones. La interacción agente-ambiente es simulada en Gazebo [86], mientras que TensorFlow [87] es empleado para la implementación de DDPG, y las redes neuronales asociadas. La interfaz entre el algoritmo de aprendizaje y las simulaciones es implementada utilizando ROS [88].

[2]Si bien otras funciones de activación para la capa de salida del crítico podrían ser empleadas (en lugar de una función de activación lineal), su utilización, en general, requeriría conocer el intervalo al que pertenecen los valores que toma la función de valor del par estado-acción.

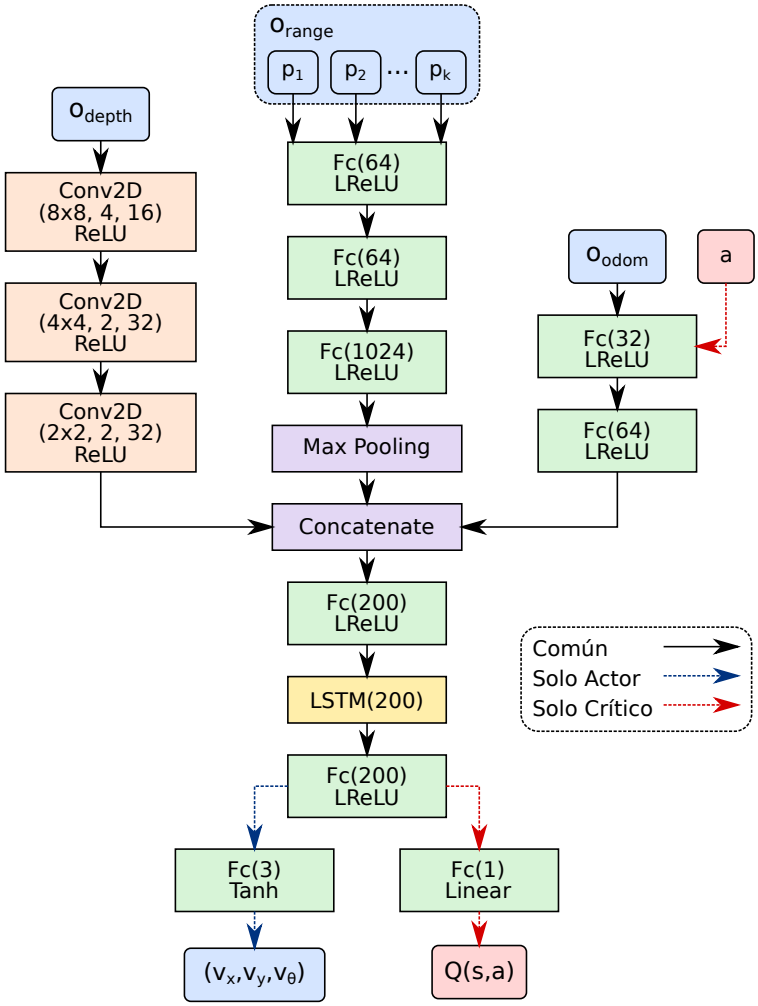


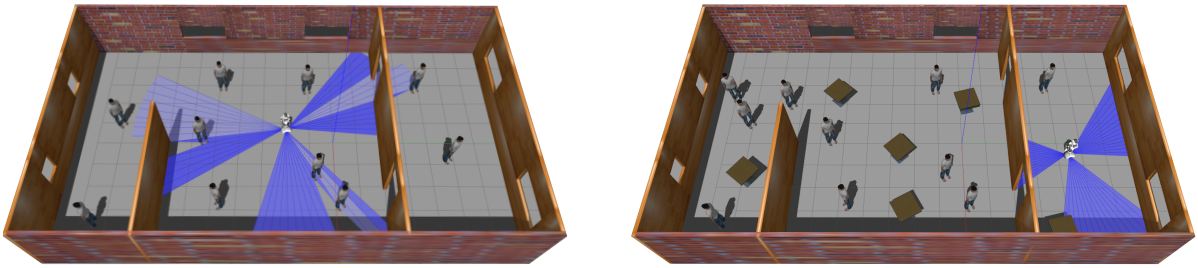
Figura 3.5: Parametrización multimodal para el actor y el crítico. Las capas son descritas siguiendo la notación "Tipo (parámetros) Función de activación", o simplemente "Tipo", en el caso de capas no paramétricas. "Fc" significa "*Fully connected*" y su parámetro representa el número de unidades ocultas asociado a la capa (esto también aplica para la capa LSTM). "Conv2D" significa "Convolución 2D", sus parámetros corresponden a las dimensiones del *kernel*, su *stride* y al número de *kernels* empleados. Finalmente, "LReLU" representa al término "Leaky ReLU".

Por otra parte, todo el procesamiento computacional requerido es realizado en un ordenador portátil equipado con un procesador Intel i7-7700HQ, y una unidad de procesamiento gráfico Nvidia GeForce GTX 1060.

3.4.1. Entrenamiento y evaluación en simulaciones

El entrenamiento es conducido en un ambiente simulado con características similares a las que presentan los espacios de interior. Este ambiente consiste en una habitación de 18 metros de largo por 10 metros de ancho, que cuenta con paredes exteriores e interiores, las cuales poseen cavidades que se asemejan a puertas y ventanas. En regiones no ocupadas de esta habitación, diferentes obstáculos son ubicados.

Utilizando este ambiente, dos escenarios son generados (ver Figura 3.6): Un escenario en el que los obstáculos adicionales dispuestos en la habitación corresponden a personas, al que se hará referencia como GWS (ver Figura 3.6a), y un escenario donde los obstáculos incluyen a personas y mesas, al que se hará referencia como GWC (ver Figura 3.6b).



(a) Gazebo World Simple (GWS) **(b)** Gazebo World Complex (GWC)

Figura 3.6: Escenarios simulados en Gazebo para entrenar y evaluar a las políticas de evasión de colisiones. (a) Contiene 10 obstáculos simples (personas). (b) Contiene 10 obstáculos simples (personas), y 6 obstáculos complejos (mesas). En ambos escenarios es posible visualizar al robot Pepper simulado. La región azul en torno a su base representa el rango perceptual que le entregan los LiDARs empleados para la construcción de sus observaciones (ver Sección 3.3.3 y Figura 3.3). Obtenido de [25].

Utilizando estos escenarios, el entrenamiento de las políticas es realizado bajo una formulación episódica: cada episodio tiene una duración máxima de 500 interacciones agente-ambiente, y puede acabar prematuramente si el robot colisiona. Además, al inicio de cada episodio, tanto la pose del robot como la de los obstáculos no fijos (personas y/o mesas) es aleatorizada. En todos los casos, tanto las mediciones de rango provenientes de los LiDARs empleados para la construcción de mapas locales, como las estimaciones de velocidad que son recibidas por el agente, son ideales.

Para todos los experimentos realizados, los parámetros empleados para el entrenamiento de las políticas se mantienen constantes, y se resumen en la Tabla 3.1. Los parámetros no reportados para DDPG son directamente tomados de [34].

	Parámetro	**Valor**
DDPG	Tasa de aprendizaje del actor	0.0001
	Tasa de aprendizaje del crítico	0.001
	Factor de suavizado λ	0.001
	Factor de descuento γ	0.99
	Tamaño máximo del *Experience Replay*	100000
	Tamaño del *mini batch*	64
Control	Frecuencia de control (Hz)	2.5
	$[v_x^{\min}, v_x^{\max}]$ (m/s)	[-0.5, 0.5]
	$[v_y^{\min}, v_y^{\max}]$ (m/s)	[-0.3, 0.3]
	$[v_\theta^{\min}, v_\theta^{\max}]$ (rad/s)	[-0.5, 0.5]

Tabla 3.1: Parámetros para el entrenamiento de las políticas de evasión de colisiones.

Durante el entrenamiento, 50 episodios de evaluación son conducidos cada una cierta cantidad de interacciones agente-ambiente. En estos episodios de evaluación, el ruido de exploración es

deshabilitado. Con el objetivo de medir la evolución del desempeño de las políticas entrenadas, dos métricas de desempeño son consideradas, y calculadas durante estas evaluaciones:

- Retorno promedio: Se calcula como el promedio de los retornos no descontados obtenidos por el agente durante los episodios en que es evaluado (en este caso, 50).
- Velocidad de avance promedio: Se define como el promedio de la velocidad instantánea en dirección x a la que el agente se mueve durante los episodios de evaluación, considerando un factor de penalización para velocidades angulares altas. Formalmente, se calcula promediando los valores que toma la expresión $\frac{1}{T}\sum_{t=1}^{T} v_x^t \cos(v_\theta^t)$ para cada episodio (en este caso, 50), donde T corresponde a la cantidad de pasos de tiempo en que el agente interactúa con el ambiente antes de llegar a un estado terminal (por colisionar, o porque el límite de pasos de tiempo es alcanzado).

Mediciones de rango versus mapa local

Con el fin de evaluar las ventajas asociadas a la utilización de la representación de "mapa local" que fue planteada en la Sección 3.3.3, esta es aislada de la información exteroceptiva entregada por las imágenes de profundidad, y comparada con una representación vectorial clásica para mediciones de rango.

Formalmente, en este experimento las observaciones de los agentes entrenados son re-definidas como $(o^t_{\text{range}}, o^t_{\text{odom}})$, donde o^t_{range} es representada como un "mapa local", o simplemente como un vector de mediciones normalizadas de rango. Consecuentemente, la parametrización original de la política (presentada en la Sección 3.3.4) es modificada para compatibilizar con cada nueva formulación, lo que deriva en dos parametrizaciones diferentes: una equivalente a la ilustrada en la Figura 3.5, omitiendo el extractor de características asociado a o^t_{depth}, y otra que también omite el extractor de características asociado a o^t_{depth} y que procesa las mediciones normalizadas de rango mediante una secuencia de capas *fully connected*.

Los agentes estudiados en este experimento son entrenados en GWS (ver Figura 3.6a), considerando que en este escenario la percepción 2D es suficiente para caracterizar a los obstáculos fijos (paredes internas y externas), y a los obstáculos cuya pose es aleatorizada en cada episodio (personas). Para ambos agentes, el entrenamiento es conducido por $7.2{\cdot}10^4$ pasos de tiempo (proceso que toma cerca de 10 horas por cada instancia entrenada), siendo los episodios de evaluación realizados cada $8{\cdot}10^3$ pasos de tiempo.

Para obtener una medida significativa del desempeño de los agentes estudiados, cada uno es entrenado diez veces de forma independiente. La Figura 3.7 muestra los resultados obtenidos para este experimento (promedio y desviación estándar de las métricas obtenidas considerando a todas las instancias entrenadas por agente).

Los resultados obtenidos muestran una clara diferencia en el desempeño general de los dos tipos de agente estudiados: los agentes que representan a o^t_{range} como un vector de mediciones normalizadas de rango, alcanzan un desempeño muy pobre en comparación a los agentes que representan a o^t_{range} a través de mapas locales.

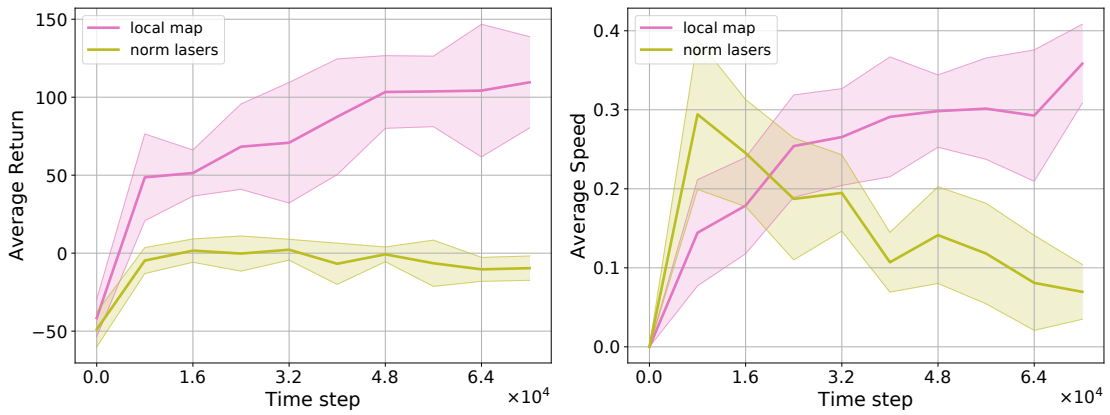


Figura 3.7: Evolución del desempeño de las políticas de evasión de colisiones con percepción 2D al ser entrenadas en GWS. (a) Retorno promedio, y (b) velocidad de avance promedio. Adaptado de [25].

Considerando que las únicas diferencias entre estos dos tipos de agentes son la formulación de sus observaciones y las parametrizaciones de sus políticas, es claro que la representación de mapas locales propuesta presenta propiedades que son útiles para este caso de estudio. En particular, los mapas locales permiten aliviar las limitaciones perceptuales del robot Pepper, actuando como una memoria explícita que permite representar regiones fuera de su campo de visión instantáneo. En contraste, al emplear mediciones normalizadas de rango, existen regiones ciegas en el campo de visión del robot, lo que limita fuertemente la capacidad de poder caracterizar apropiadamente el ambiente.

Dotando al agente de percepción 3D

Con el fin de evaluar el desempeño del sistema propuesto en su forma completa, en este caso se entrenan agentes que incluyen a la componente o^t_{depth} como parte de sus observaciones, y cuyas políticas son parametrizadas de acuerdo a lo expuesto en la Sección 3.3.4, sin realizar ninguna alteración a la arquitectura mostrada en la Figura 3.5.

A diferencia del experimento anterior, en este caso el entrenamiento es conducido en GWC (ver Figura 3.6b), donde se requiere que los agentes cuenten con percepción 3D para poder evadir a las mesas, cuyas geometrías de colisión no pueden ser estimadas de forma apropiada si solo se emplean mediciones de rango 2D.

Bajo estas condiciones, el entrenamiento es realizado durante $1.2{\cdot}10^5$ pasos de tiempo (proceso que en este caso toma cerca de 15 horas por cada instancia entrenada), mientras que los episodios de evaluación toman lugar cada 10^4 pasos de tiempo. Con el fin de obtener medidas significativas del desempeño del sistema propuesto, este es entrenado diez veces. La Figura 3.8 muestra los resultados obtenidos (promedio y desviación estándar considerando a todas las instancias entrenadas).

Los resultados muestran que los agentes entrenados empleando la formulación propuesta logran alcanzar buenos desempeños de forma consistente. En particular, se logra validar la efectividad del

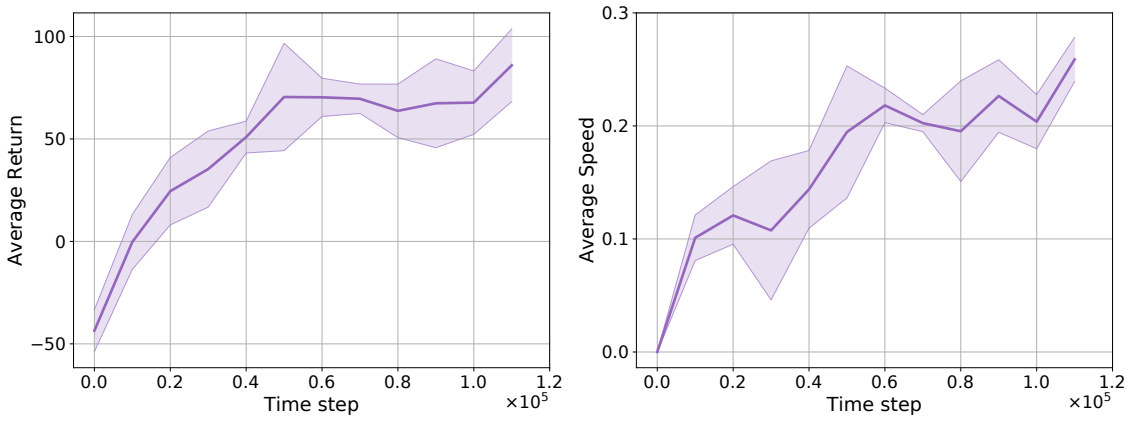


Figura 3.8: Evolución del desempeño de la política propuesta al ser entrenada en GWC. (a) Retorno promedio, y (b) velocidad de avance promedio. Adaptado de [25].

uso de la paramatrización propuesta como una forma simple de combinar mediciones provenientes de diferentes fuentes de información. Para lograr la evasión de mesas, por ejemplo, es necesario que el agente aprenda a interpretar mediciones aparentemente contradictorias, puesto que los mapas locales las detectan como "puntos" o regiones libres de obstáculos, y solo las imágenes de profundidad logran caracterizaras apropiadamente.

Hay que notar, además, que los resultados obtenidos en este experimento muestran que existe una ligera disminución de desempeño al comparar al sistema completo con la variante que solo usa mapas locales. Esta disminución de rendimiento se ve reflejada en valores promedio de retornos y velocidades de avance menores, no obstante, la desviación estándar de estas métricas es también menor en este experimento.

Es posible dar una explicación a este resultado al considerar las diferencias de complejidad entre los escenarios GWS y GWC. En GWS, los únicos obstáculos cuya pose es aleatorizada para cada episodio corresponden a 10 personas. En GWC, en contraste, los obstáculos cuya pose es aleatorizada en cada episodio corresponden a 10 personas y 6 mesas. En GWS, por tanto, pueden existir configuraciones específicas donde la navegación del agente por el ambiente es relativamente simple, lo que se traduce en la obtención de métricas con mejores promedios y mayor variabilidad. Al considerar un ambiente más complejo debido a una mayor cantidad de obstáculos, como GWC, la dificultad de los episodios aumenta, pero además, esta dificultad se vuelve más homogénea entre episodios diferentes, pues dejan de existir configuraciones para la pose de los obstáculos que faciliten la navegación del agente en el ambiente. Esto, en consecuencia, resulta en métricas de desempeño con valores promedio y desviaciones estándar inferiores, que corresponde a lo observado en este experimento.

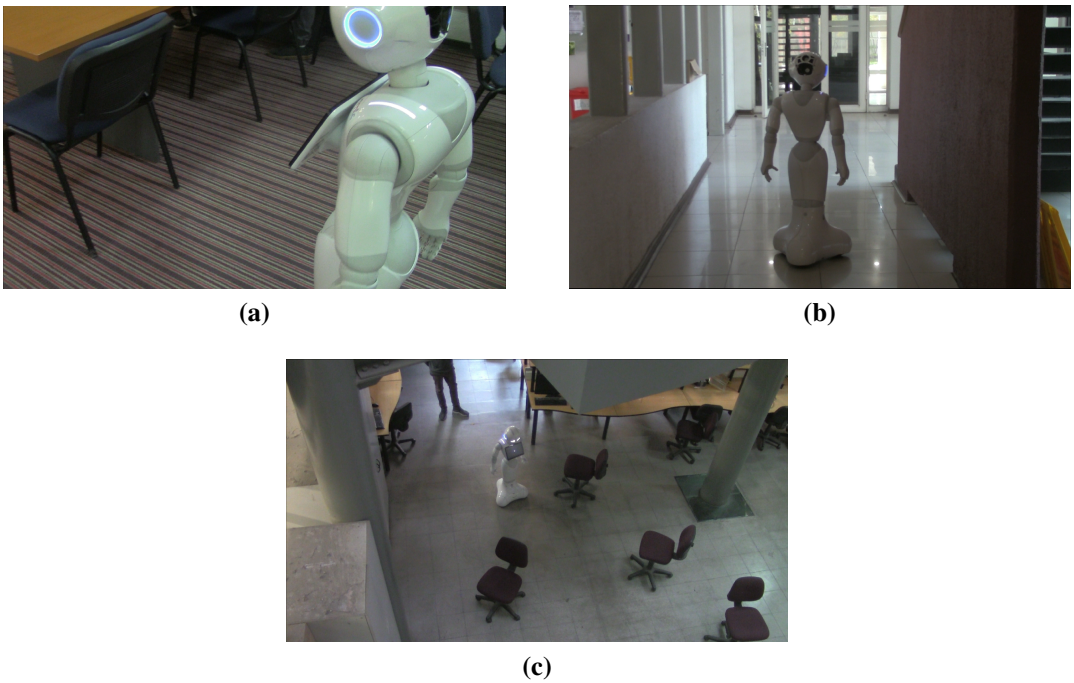
3.4.2. Validación en el mundo real

Con el objetivo de evaluar la aplicabilidad del enfoque propuesto en el mundo real, una instancia de las políticas entrenadas en el escenario GWC es desplegada en diferentes ambientes reales. La

transferencia de la política, desde simulaciones a la realidad, es realizada de forma directa mediante el uso de procesamiento externo e interfaces provistas por ROS.

Las imágenes de profundidad reales obtenidas por el sensor de profundidad de Pepper son procesadas de acuerdo a la descrito en la Sección 3.3.2, mientras que la construcción de los mapas locales es realizada al igual que en simulaciones, según lo expuesto en la Sección 3.3.3.

Para evaluar el desempeño del sistema propuesto y, en particular, su capacidad de generalización, este es desplegado en tres ambientes de interior diferentes (ver Figura 3.9): una sala de estudio con obstáculos estáticos, un vestíbulo con obstáculos estáticos y dinámicos, y la sala de un laboratorio que cuenta con obstáculos dispuestos de forma laberíntica. Algunos ejemplos del comportamiento mostrado por la política en estos ambientes, pueden ser visualizados en `https://youtu.be/ypC39m4BlSk`.



(a) **(b)** **(c)**

Figura 3.9: Ambientes reales empleados para la validación del sistema de evasión de colisiones propuesto. (a) Corresponde a una sala de estudio, (b) a un vestíbulo, y (c) a la sala de un laboratorio. Obtenido de [25].

Los experimentos conducidos en ambientes reales muestran que el sistema propuesto es capaz de dotar al robot Pepper con la capacidad de evadir colisiones de manera efectiva. En todos los ambientes considerados, el robot es capaz de navegar por regiones despejadas de forma autónoma, evadiendo colisiones contra obstáculos estáticos y dinámicos. Este resultado demuestra la gran capacidad de generalización de la política entrenada, considerando que los ambientes reales de prueba son bastante diferentes a GWC.

El éxito en el despliegue de la política en el mundo real puede ser explicado considerando dos factores. En primer lugar, en la mayoría de las situaciones experimentadas por el agente, los mapas locales construidos con datos reales son similares a aquellos construidos con datos simulados. En segundo lugar, para lograr la evasión de obstáculos que son percibidos principalmente por la cámara de profundidad, información de baja granularidad es suficiente. Con esto, es posible explicar que

el agente logre comportarse apropiadamente incluso cuando las observaciones que recibe distan de aquellas que fueron experimentadas durante el proceso de entrenamiento. Esto ocurre, por ejemplo, cuando el sensor de profundidad percibe objetos arbitrarios, no considerados en simulaciones.

Una limitación importante del sistema, es que depende fuertemente de la calidad de las imágenes de profundidad empleadas para la construcción de las observaciones del agente. Experimentalmente se observa que, en los casos en que las imágenes de profundidad presentan distorsiones extremas (por ejemplo, cuando el sistema se despliega en exteriores, debido a la presencia de luz natural), el agente adopta comportamientos poco predecibles. Esto ocurre principalmente debido a que, en estos casos, el método de reconstrucción aplicado sobre las imágenes no cuenta con suficiente información con la cual generar representaciones apropiadas. Dependiendo de la información que exista en el mapa de profundidad original, esto puede ocasionar la generación de imágenes que visualmente pueden ser interpretadas como regiones completamente libres, o como obstáculos próximos, ocasionando que el agente ejecute acciones basadas en información errónea.

3.5. Discusión

En este capítulo se ha presentado una propuesta que permite la obtención de una política de evasión de colisiones usando aprendizaje reforzado. La política es parametrizada mediante una red neuronal multimodal, y permite mapear imágenes de profundidad, estimaciones de velocidad basadas en odometría y nubes de puntos 2D construidas a partir de mediciones de rango, a comandos de velocidad para un robot omnidireccional.

Como caso de estudio, el robot Pepper fue utilizado como plataforma física. Extensivas pruebas experimentales permitieron demostrar la efectividad del enfoque propuesto. En particular, se encontró que el uso de nubes de puntos como representación para las observaciones del agente es una alternativa viable, y que, dadas sus características, permite mejorar artificialmente las capacidades perceptuales del agente cuando se construye a partir de la integración de mediciones de rango instantáneas. Adicionalmente, se verificó que la transferencia de la política entrenada, de simulaciones al mundo real, es factible considerando un pre-procesamiento sobre las imágenes de profundidad.

Al desplegar el sistema de evasión de colisiones desarrollado en el mundo real, se comprobó que un desempeño aceptable es alcanzado a pesar de considerar ambientes radicalmente diferentes a los utilizados durante el proceso de entrenamiento. Lo anterior muestra la gran capacidad de generalización de las políticas entrenadas.

Experimentalmente se observó que una limitación del sistema propuesto, es que su desempeño se ve perjudicado en casos en que las imágenes de profundidad empleadas como parte de las observaciones del agente presentan distorsiones significativas, por ejemplo, debido a la presencia de luz natural en el ambiente. Una forma de soslayar este problema podría ser la utilización de representaciones alternativas, en reemplazo a los mapas de profundidad utilizados, o la incorporación de estrategias para hacer que la política multimodal sea robusta a esta clase de perturbaciones, como por ejemplo, "*sensor dropout*" [74].

Otra limitación del sistema desarrollado, es que la integración de las mediciones de rango para la

construcción de las nubes de puntos usadas como observaciones, se basa en el uso de información de odometría. Esto limita la calidad de la integración en el mundo real, por lo que métodos más sofisticados para registrar las mediciones (e.g. *Iterative Closest Point*) podrían ser empleados para mejorar los mapas locales.

Finalmente, este caso de estudio solo abordó el problema de evasión de colisiones. Este problema, por su naturaleza, es uno de los más simples en el espectro de sub-problemas asociados a navegación robótica. Considerando los resultados obtenidos, y la extensibilidad de la formulación propuesta a tareas diferentes, resulta interesante cuestionar si su aplicabilidad es viable en problemas diferentes, de mayor complejidad. Con el fin de dar una respuesta parcial a esta pregunta, en el capítulo siguiente se aborda el problema de planificación local.

Capítulo 4

Planificación local usando nubes de puntos 2D como observaciones

En este capítulo se aborda el problema de planificación local para robots móviles usando aprendizaje reforzado. Concretamente, se presenta una propuesta que permite la obtención de una política parametrizada mediante una red neuronal artificial, la cual mapea observaciones a acciones continuas para el control de un robot diferencial.

Extendiendo el trabajo presentado en el capítulo anterior, parte de las observaciones del agente son representadas a través de nubes de puntos 2D de tamaño variable. Los beneficios relacionados al uso de esta representación son justificados de forma detallada, y se realiza una comparación entre su utilización, y la utilización de otras representaciones que han sido propuestas en trabajos previamente publicados.

A través de evaluaciones en simulaciones, se verifica experimentalmente que el enfoque propuesto permite el entrenamiento de políticas robustas a perturbaciones extremas en sus observaciones. Más aún, se verifica que las políticas desarrolladas son directamente transferibles desde simulación a la realidad, y que son integrables a un sistema de navegación completo, donde adicionalmente es posible mejorar las capacidades perceptuales del agente "sobre la marcha", mediante técnicas simples de pre-procesamiento y/o fusión de datos sensoriales.

4.1. Motivación

Como se señaló en el Capítulo 1, los sistemas clásicos de navegación suelen caracterizarse por ser modulares: cada módulo aborda un sub-problema del problema completo asociado a la navegación autónoma, e interactúa con los otros módulos. Un sistema clásico de navegación, por ejemplo, puede componerse de un módulo encargado de la localización del robot, un módulo encargado de la generación de planes globales hacia el objetivo de navegación, y un módulo encargado del seguimiento de dichos planes. En este contexto, la planificación local puede definirse como el problema de lograr alcanzar objetivos locales de navegación, a partir de información sensorial instantánea (local) del ambiente. Este es un problema más complejo que el de evasión de colisiones, pues no

solo se requiere que el agente evite colisionar contra obstáculos mientras se desplaza, sino también que su desplazamiento este dirigido a un objetivo específico.

En los últimos años, un gran número de trabajos abordando el problema de planificación local usando metodologías basadas en aprendizaje han sido propuestos. El estudio de esta clase de sistemas es motivado por las propiedades deseables que estos presentan: son por naturaleza reactivos, y consecuentemente, adecuados para ser desplegados en ambientes dinámicos. Considerando estos antecedentes, resulta interesante comprobar si el uso de representaciones similares a los mapas locales utilizados en el capítulo anterior, podría ser útil en la tarea de planificación local.

Mientras que el uso de diferentes fuentes de información exteroceptiva es efectivo en ciertos contextos (como en el caso de estudio previo), en este caso el enfoque es puesto exclusivamente en la utilización de mediciones de rango. Considerando las limitaciones perceptuales que presenta el robot Pepper (en particular, aquellas asociadas a los LiDAR cercanos a su base, y descritas en la Sección 3.3.3), en este caso de estudio, un robot Pioneer 3-DX equipado con sensores de rango es empleado. Lo anterior, para facilitar las comparaciones con un espectro más amplio de trabajos previos en función de las parametrizaciones, y las codificaciones para las observaciones que son empleadas; y consecuentemente, permitir un análisis más claro en relación al impacto que estas decisiones de diseño generan.

4.2. Trabajos relacionados

El desarrollo de sistemas de navegación robótica mediante métodos basados en aprendizaje se ha popularizado en los últimos años. Varios trabajos han reportado el desarrollo de sistemas desplegables en robots reales, correspondiendo estos sistemas típicamente a políticas de planificación local (e.g. [23, 89, 77, 24]).

El entrenamiento de estas políticas puede ser realizado en el marco de trabajo del aprendizaje supervisado: mediante la utilización de datos etiquetados, es posible aprender un mapeo entre observaciones y acciones que permita a un agente alcanzar un determinado objetivo de navegación, evitando colisiones. Trabajos siguiendo esta metodología han logrado desarrollar planificadores locales efectivos (e.g. [90, 89]), no obstante, y como se discutió en la Sección 3.2, este enfoque requiere la obtención de una base de datos con ejemplos representativos, cuyas etiquetas permitan el aprendizaje de comportamientos deseados. La obtención de una base de datos con estas características usualmente es una tarea difícil.

Los planificadores locales desarrollados utilizando aprendizaje reforzado, en contraste, son entrenados a partir de la experiencia que se genera a través de interacciones entre el agente y su ambiente. Este entrenamiento comúnmente es conducido en simulaciones debido a que, por lo general, una gran cantidad de interacciones son necesarias para lograr obtener una política efectiva.

Si bien en algunos casos de estudio se ha logrado el desarrollo de políticas de navegación visual (e.g. [91, 92, 77]), su despliegue en el mundo real es desafiante debido a las diferencias existentes entre imágenes simuladas, e imágenes capturadas con un sensor real. Enfoques más pragmáticos para el desarrollo de políticas de planificación local, tienden a caracterizar el ambiente mediante el

uso de sensores de rango. Una de las ventajas de esta decisión de diseño, es que el problema asociado a la brecha simulación-realidad es más fácilmente abordable, o incluso despreciable. Ejemplos de aplicaciones exitosas que siguen esta metodología, incluyen el desarrollo de planificadores locales cuyas observaciones corresponden a mediciones de rango representadas mediante vectores de baja dimensionalidad [23, 24, 93]. En estos casos, estos vectores son utilizados como entradas a parametrizaciones simples de las políticas, las cuales consisten en redes neuronales *feed forward*. La principal desventaja asociada al uso de esta clase de observaciones, es que pueden no ser suficientes para caracterizar al ambiente de forma apropiada. En escenarios complejos, por ejemplo, esta clase de representaciones pueden no proveer una observabilidad suficientemente buena como para ejercer control de alta granularidad sobre el agente.

Con el propósito de aliviar la observabilidad parcial del ambiente, algunos trabajos han propuesto el uso de "*stacks*" de mediciones de rango [94, 95, 96, 97]. Esta técnica consiste en la agregación de mediciones en el tiempo en una cola de tamaño fijo, de modo que la política incorpore información instantánea y pasada para generar acciones de control. Mientras que en [95] estas observaciones son procesadas mediante el uso de capas *fully connected*, en [94, 96] y [97] son procesadas usando capas convolucionales.

Alternativamente, la utilización de redes recurrentes para la parametrización de la política tambien ha sido una propuesta para aliviar la observabilidad parcial del ambiente [91, 77, 96]. La idea es aprender una representación más completa del estado del ambiente a partir del uso de información histórica de las interacciones agente-ambiente. Esta información es almacenada en una memoria interna, implementada mediante, por ejemplo, capas LSTM.

El caso de estudio presentado en este capítulo se relaciona directamente con el cuerpo de trabajos sobre planificadores locales basados en RL, en los que las observaciones del agente son codificadas mediante representaciones de tamaño fijo. En este sentido, se busca mostrar cuales son los beneficios asociados al reemplazo de esta codificación clásica para las observaciones del agente, por nubes de puntos de tamaño variable.

Esta clase de representaciones ha sido empleada en trabajos previos, particularmente, en el caso de estudio que fue presentado en el capítulo anterior [25], pero también en el contexto de robótica de enjambres [98], donde es relevante poder caracterizar a un número variable de agentes que interactúan con un agente en particular. A diferencia de estos trabajos, no obstante, en este caso de estudio se busca validar la efectividad de esta representación al abordar el problema de planificación local, el cual es más complejo que el de evasión de colisiones, y donde el uso de esta representación tiene justificaciones más practicas que en el caso de robótica de enjambres.

4.3. Propuesta

4.3.1. Modelamiento del problema

Al igual que para el modelamiento del problema de evasión de colisiones presentado en el Capítulo 3, la interacción agente-ambiente es modelada como un Proceso de Decisión de Markov Parcialmente Observable (POMDP, ver Sección 2.1.1). En consecuencia, las observaciones $o_t \in \Omega$

y acciones $a_t \in \mathcal{A}$ son diseñadas considerando la percepción y actuación del agente, y la función de recompensa r es diseñada para guiar el aprendizaje de la política de planificación local deseada.

Observaciones

Las observaciones recibidas por el agente en cada instante de tiempo se definen como $o_t = (o^t_{\text{pcl}}, o^t_{\text{odom}}, o^t_{\text{target}})$, donde o^t_{pcl} corresponde a la nube de puntos que resulta del procesamiento de las mediciones de rango obtenidas por el robot, o^t_{odom} a estimaciones de su velocidad, basadas en odometría, y o^t_{target} al objetivo de navegación en coordenadas polares.

La nube de puntos o^t_{pcl} se construye de manera similar a o^t_{range} (ver Sección 3.3.3), no obstante, a diferencia de o^t_{range}, solo contiene información instantánea. Una descripción (y justificación) detallada de o^t_{pcl} se presenta en la Sección 4.3.2. La componente o^t_{odom} queda definida por la tupla $(\hat{v}^t_x, \hat{v}^t_\theta)$, donde $\hat{v}^t_x$ corresponde a la estimación de velocidad lineal del robot en el eje x, y $\hat{v}^t_\theta$ a la estimación de velocidad angular del robot en torno al eje z. Finalmente, la componente $o^t_{\text{target}} = (\rho^t_{\text{target}}, \theta^t_{\text{target}})$ corresponde al objetivo de navegación en coordenadas polares, con respecto al sistema de coordenadas local del robot.

Acciones

Las acciones corresponden a comandos de velocidad continuos para un robot diferencial. Una acción dada, para el instante de tiempo t, es definida como $a_t = (v^t_x, v^t_\theta)$, donde $v^t_x \in [v^{\text{min}}_x, v^{\text{max}}_x]$ es la velocidad lineal del robot en el eje x, y $v^t_\theta \in [v^{\text{min}}_\theta, v^{\text{max}}_\theta]$ es su velocidad angular en torno al eje z. En este caso, los límites para v^t_x son tales que $v^{\text{min}}_x = 0$ y $v^{\text{max}}_x > 0$, mientras que los límites para v^t_θ son tales que $v^{\text{max}}_\theta > 0$, y $v^{\text{min}}_\theta = -v^{\text{max}}_\theta$ (los valores numéricos para los límites señalados son reportados en la Tabla 4.1).

Recompensa

La función de recompensa es definida de acuerdo a (4.1), donde ρ^t_{target} es la coordenada radial de o^t_{target} (distancia euclidiana entre la posición del robot y el objetivo de navegación), y $\rho^{\text{thresh}}_{\text{target}}$ es una distancia fija, usada como umbral para determinar si el robot ha alcanzado el objetivo de navegación.

$$r_t = \begin{cases} r^t_{\text{navigation}} & \text{si } \rho^t_{\text{target}} \geq \rho^{\text{thresh}}_{\text{target}} \\ r^t_{\text{success}} & \text{si } \rho^t_{\text{target}} < \rho^{\text{thresh}}_{\text{target}} \\ r^t_{\text{collision}} & \text{si el robot colisiona} \end{cases} \tag{4.1}$$

El término $r^t_{\text{navigation}}$ es diseñado con el fin de guiar al agente hacia el objetivo de navegación, penalizando maniobras arriesgadas y comportamientos indeseables. Este término es definido como la suma de cuatro componentes: $r^t_{\text{navigation}} = r^t_{\text{target}} + r^t_{\text{fov}} + r^t_{v_\theta} + r^t_{\text{danger}}$.

La componente r^t_{target} se define de acuerdo a (4.2)[1]. Esta componente penaliza al agente si este se aleja del objetivo de navegación, mientras que simultáneamente lo incentiva a llegar a dicho objetivo rápidamente. Puesto que $v_x^{\min} = 0$, entonces $\hat{v}_x^t / v_x^{\max} \in [0, 1]$, y como $\theta^t_{\text{target}} \in [-\pi, \pi]$, entonces el primer término que conforma a r^t_{target}, es decir, $(\hat{v}_x^t / v_x^{\max}) \cos(\theta^t_{\text{target}})$, toma valores en el intervalo $[-1, 1]$. Este término es maximizado cuando $\hat{v}_x^t = v_x^{\max}$ y $\theta^t_{\text{target}} = 0$, es decir, cuando el agente se mueve a la máxima velocidad lineal en el eje x que se le permite, y se encuentra alineado al objetivo de navegación.

$$r^t_{\text{target}} = \frac{\hat{v}_x^t}{v_x^{\max}} \cos(\theta^t_{\text{target}}) + 5 \cdot \mathbf{1}_{\{\rho^t_{\text{target}} : \rho^t_{\text{target}} < \rho^{t-1}_{\text{target}}\}}(\rho^t_{\text{target}}) - 6 \tag{4.2}$$

El segundo término que conforma a r^t_{target}, $5 \cdot \mathbf{1}_{\{\rho^t_{\text{target}} : \rho^t_{\text{target}} < \rho^{t-1}_{\text{target}}\}}(\rho^t_{\text{target}})$, recompensa al agente cada vez que este acorta su distancia al objetivo de navegación entre pasos de tiempo sucesivos. Finalmente, la constante que es restada a los dos términos antes descritos es fijada de forma tal que el valor máximo que pueda alcanzar r^t_{target} sea cero.

La componente r^t_{fov}, definida por (4.3), penaliza al agente si es que $|\theta^t_{\text{target}}|$ excede 120°, es decir, si es que el objetivo de navegación se encuentra fuera de la proyección de un FoV de 240°, medido desde el marco de referencia local del robot. Como $|\theta^t_{\text{target}}| \leq \pi$ rad, la penalización recibida por el agente cuando el término r^t_{fov} es no nulo, toma valores en el intervalo $[-8.0, -6.5)$. Al incorporar este término en la definición de $r^t_{\text{navigation}}$, se asume que los objetivos de navegación solicitados al agente no requieren que este cuente con capacidades de planificación a largo plazo para ser alcanzados.

$$r^t_{\text{fov}} = (3 \cos(\theta^t_{\text{target}}) - 5) \cdot \mathbf{1}_{\{\theta^t_{\text{target}} : |\theta^t_{\text{target}}| > 120°\}}(\theta^t_{\text{target}}) \tag{4.3}$$

La componente $r^t_{v_\theta}$, definida por (4.4) y (4.5), se encarga de penalizar velocidades y aceleraciones angulares altas. De acuerdo a (4.5), el término $K^t_{v_\theta}$ es el encargado de detectar velocidades y aceleraciones angulares altas. Como $v_\theta^{\min} = -v_\theta^{\max}$, este término toma valores en el intervalo $[0, 1]$, siendo nulo cuando $|\hat{v}^t_\theta| = 0$ y $|\hat{v}^t_\theta - \hat{v}^{t-1}_\theta| = 0$, y alcanzando su valor máximo cuando $|\hat{v}^t_\theta| = v_\theta^{\max}$, o cuando $|\hat{v}^t_\theta - \hat{v}^{t-1}_\theta| = 2v_\theta^{\max}$. Cada vez que $K^t_{v_\theta}$ es mayor a 0.5, el término $r^t_{v_\theta}$ es no nulo, y entrega una penalización al agente igual al doble del valor que tome $K^t_{v_\theta}$, es decir, en el intervalo $[-2, -1)$.

$$r^t_{v_\theta} = -2K^t_{v_\theta} \cdot \mathbf{1}_{\{K^t_{v_\theta} : K^t_{v_\theta} > 0.5\}}(K^t_{v_\theta}) \tag{4.4}$$

$$K^t_{v_\theta} = \frac{1}{2v_\theta^{\max}} \max\{2|\hat{v}^t_\theta|, |\hat{v}^t_\theta - \hat{v}^{t-1}_\theta|\} \tag{4.5}$$

Finalmente, la componente r^t_{danger}, definida por (4.6), entrega una señal de penalización cuando el agente se acerca peligrosamente a obstáculos. El valor de r^t_{danger} depende de $l^t_{\min}$, que corresponde a la mínima distancia medida por el sensor de rango del robot, en metros. Notando que la expresión $60 \max\{l^t_{\min} - 0.35, 0\} \in [0, 3)$ cuando $l^t_{\min} < 0.4$, es claro que cuando el término r^t_{danger} es no nulo, toma valores en el intervalo $[-5, -2)$, alcanzando su valor mínimo para $l^t_{\min} \leq 0.35$.

$$r^t_{\text{danger}} = (60 \max\{l^t_{\min} - 0.35, 0\} - 5) \cdot \mathbf{1}_{\{l^t_{\min} : l^t_{\min} < 0.4\}}(l^t_{\min}) \tag{4.6}$$

[1]Para un conjunto A, la función indicatriz $\mathbf{1}_B(x)$, donde $B \subseteq A$, es igual a 1 si $x \in B$, e igual a 0 en caso contrario.

Los valores numéricos del umbral de distancia empleado para que r^t_{danger} sea no nulo (0.4), y el de la constante utilizada para fijar su valor mínimo (0.35), son elegidos tomando en cuenta la geometría aproximada de la huella (*footprint*) del robot utilizado (el Pioneer 3-DX). Naturalmente, para robots con *footprints* cuyas geometrías fuesen más complejas, el término r^t_{danger} debería ser re-diseñado para poder detectar apropiadamente cuando el agente se acerque peligrosamente a obstáculos.

Los términos r^t_{success} y $r^t_{\text{collision}}$, solo son recibidos por el agente cuando este alcanza estados terminales. El término r^t_{success} entrega una fuerte señal de recompensa al agente si este alcanza el objetivo de navegación, por lo que toma el valor de 100. En contraposición, $r^t_{\text{collision}}$ entrega una fuerte señal de penalización si este colisiona, por lo que toma el valor de -200.

Con la finalidad de fomentar que el agente llegue al objetivo de navegación tan rápido como sea posible, todos los términos que componen a $r^t_{\text{navigation}}$ poseen un valor máximo igual a cero. Adicionalmente, las constantes empleadas para la definición de estas componentes han sido cuidadosamente elegidas de modo que, en los casos en los que el agente adopta comportamientos indeseables, estos términos entregan penalizaciones con valores mínimos (aproximados) similares: -5 para r^t_{target}, -6.5 para r^t_{fov}, -1 para $r^t_{v_\theta}$ y -2 para r^t_{danger}.

4.3.2. Nubes de puntos 2D como observaciones

En este caso de estudio, la principal fuente de información para conocer el estado del ambiente con la que cuenta el agente, corresponde a un sensor de rango. Las mediciones entregadas por esta clase de sensores son fundamentales para caracterizar el entorno del agente, y de este modo, evitar colisiones al navegar en él.

La representación de mediciones de rango como vectores, ha sido ampliamente adoptada para definir las observaciones del agente al abordar el problema de navegación robótica usando aprendizaje de máquinas (e.g. [23, 89, 93]). En este contexto, la utilización de técnicas convencionales de extracción de características usando redes neuronales, impone una restricción importante sobre las observaciones: estas deben mantener un tamaño fijo, y por lo tanto, las mediciones con valores desconocidos o fuera de rango, deben ser codificadas.

Sea $x^t_{\text{range}} = [\rho^t_1, ..., \rho^t_n] \in \mathbb{R}^n$ el vector que contiene las mediciones de rango obtenidas por el agente en el tiempo t, donde ρ^t_i corresponde a la i-ésima medición, y θ_i a su ángulo correspondiente, tomando como referencia el origen del sensor. Las mediciones en x^t_{range} son tales que $\rho^t_i \in (\rho_{\text{min}}, \rho_{\text{max}}) \cup \{\underline{\rho}, \overline{\rho}\}$, donde ρ_{min} y ρ_{max} son los valores mínimo y máximo que pueden tomar las mediciones de rango, mientras que $\underline{\rho}$ y $\overline{\rho}$ son valores fijos para codificar a aquellas mediciones inferiores a ρ_{min} o superiores a ρ_{max}, respectivamente.

Por definición, el vector x^t_{range} no incorpora explícitamente la relación existente entre cada medición p^t_i y su angulo correspondiente, θ_i. Esta relación debe ser inferida durante el entrenamiento, por lo que x^t_{range} debe mantener una distribución geométrica constante de las mediciones p^t_i en cada instante de tiempo: la i-ésima medición siempre debe estar asociada al ángulo θ_i, el cual determina el índice de p^t_i en el vector x^t_{range}.

En el caso en que se empleen técnicas de submuestreo para reducir el número de mediciones incluído en x^t_{range}, también es necesario que las mediciones seleccionadas mantengan una distribución geométrica constante. Lo anterior es cierto incluso cuando el submuestreo es realizado agregando mediciones bajo algún criterio (e.g. rango mínimo [24] o rango promedio [93]), solo que en este caso las mediciones incluidas en x^t_{range} quedan asociadas a intervalos angulares, en lugar de ángulos únicos.

Considerando lo anterior, se propone una representación alternativa a x^t_{range}, que incorpore de forma explícita la información angular asociada a cada medición de rango, y que elimine la necesidad de que estas mantengan una distribución geométrica constante. La representación propuesta corresponde a la nube de puntos definida por (4.7), donde $k \geq 0$ corresponde al número de mediciones dentro del intervalo $(\rho_{\text{min}}, \rho_{\text{max}})$, es decir, al número de mediciones "*in-range*".

$$o^t_{\text{pcl}} = \begin{cases} \{(\rho^t_j \cos(\theta_j), \rho^t_j \sin(\theta_j))\}^{k \leq n}_{j=1} & \text{si } k \geq 1 \\ \{(\rho_{\text{max}}, 0)\} & \text{si } k = 0 \end{cases} \tag{4.7}$$

Esta representación presenta ventajas prácticas al ser comparada con su contraparte vectorial, pues separa las características del sensor empleado de la representación de sus mediciones. La incorporación explícita de información angular permite, hipotéticamente, el aprendizaje de representaciones intermedias que son independientes de la distribución geométrica de las mediciones. Esto permitiría, por ejemplo, variaciones en el campo de visión del robot. Adicionalmente, las representaciones intermedias que se aprenderían también serían independientes de la resolución del sensor, pues o^t_{pcl} es un conjunto de puntos de cardinalidad variable. Esto permitiría, por ejemplo, realizar submuestreo o integración de mediciones en línea, como en la construcción de los mapas locales descritos en la Sección 3.3.3.

4.3.3. Algoritmo y parametrización de la política

Para entrenar la política de planificación local, el algoritmo DDPG [34] es utilizado (ver Sección 2.3.3), por lo que dos redes neuronales independientes parametrizan a la política (actor) y a la función de valor del par estado-acción (crítico), respectivamente.

Al igual que para la evasión de colisiones usando al robot Pepper (ver Sección 3.3.4), las observaciones del agente, en este caso de estudio, son representadas utilizando estructuras de datos heterogéneas. En efecto, estas se definen como $o_t = (o^t_{\text{pcl}}, o^t_{\text{odom}}, o^t_{\text{target}})$, donde la componente o^t_{pcl} es representada como una nube de puntos (ver Sección 4.3.2), mientras que tanto o^t_{odom} como o^t_{target} son representadas como vectores. En consecuencia, se emplea un enfoque multimodal para el procesamiento de la información contenida en las observaciones del agente, como en la Sección 3.3.4.

Dado que o^t_{pcl} es una nube de puntos de tamaño variable, la misma estrategia descrita en la Sección 3.3.4 es utilizada para procesarla, es decir, se utiliza un módulo de extracción de características como el propuesto en PointNet [35] (cada punto de o^t_{pcl} es normalizado y procesado por la misma secuencia de capas *fully connected*, y las salidas son agregadas usando *max pooling*). El procesamiento de los vectores o^t_{odom} y o^t_{target}, por otra parte, es realizado de forma convencional, siendo estos normalizados y usados como entradas a secuencias de capas *fully connected* independientes.

Si bien tanto o^t_{odom} como o^t_{target} son vectores, la separación de su procesamiento es realizada con el fin de balancear las dimensiones de las representaciones intermedias asociadas a cada componente de o_t, de modo que su influencia en el comportamiento del agente sea comparable.

Las arquitecturas de las parametrizaciones para las redes neuronales asociadas al actor y al crítico, se resumen en la Figura 4.1. Al igual que en las arquitecturas presentadas en la Figura 3.5, las salidas de los extractores de características de cada componente de las observaciones son concatenadas, y procesadas por una nueva secuencia de capas. La utilización de las funciones sigmoide y tanh como funciones de activación para las capas de salida del actor, permite acotar los valores numéricos de las componentes de las acciones asociadas a la velocidad lineal y angular a los intervalos $[0, 1]$ y $[-1, 1]$, respectivamente. Lo anterior permite escalar fácilmente los valores que toman estas salidas a los intervalos definidos por los límites $v_x^{\min}$, $v_x^{\max}$, $v_\theta^{\min}$ y $v_\theta^{\max}$ (especialmente considerando que para este caso de estudio $v_x^{\min} = 0$). Por otra parte, la función de activación asociada a la capa de salida del crítico es lineal por el motivo anteriormente expuesto en la Sección 3.3.4, es decir, debido a que se desea aproximar una función que toma valores reales en un cierto intervalo.

La Figura 4.1 muestra dos variantes de la parametrización propuesta: la parametrización PCL-LSTM cuenta con una capa LSTM, mientras que la parametrización PCL no. Al igual que para el sistema de evasión de colisiones propuesto en el Capítulo 3, las capas LSTM son incluídas en las parametrizaciones para el actor y el crítico con el fin de aliviar la observabilidad parcial del ambiente. Consecuentemente, la actualización de los pesos de las redes neuronales que incluyen este tipo de capas, es realizado siguiendo el enfoque utilizado en [77] y validado en [85].

4.4. Evaluación experimental

La interacción agente-ambiente es simulada en Stage [99], mientras que TensorFlow [87] es empleado para la implementación de DDPG y las redes neuronales asociadas. Por otra parte, la validación del sistema es realizada tanto en simulaciones, utilizando Gazebo [86], como en el mundo real. Al igual que en el caso de estudio presentado en el Capítulo 3, la interfaz entre el algoritmo de aprendizaje y las simulaciones es implementada usando ROS [88].

Para la validación en el mundo real se utiliza el robot Pioneer 3-DX, un robot móvil diferencial ampliamente utilizado como plataforma de investigación. Este robot es equipado con un LiDAR Hokuyo URG-04LX-UG01, el cual posee un rango máximo de medición de 5.6 metros, una resolución angular de aproximadamente 0.352°, y un FoV de 240°. Adicionalmente, el robot es equipado con una cámara RGB-D Asus Xtion alineada con el eje x del sistema de coordenadas local del robot. Esta cámara es dispuesta en el robot con el fin de mejorar sus capacidades de percepción en el mundo real. Consecuentemente, los robots simulados en Stage y Gazebo, son construidos para que se asemejen al robot real empleado.

Todo el procesamiento computacional requerido para el entrenamiento y validación de las políticas en simulaciones, es realizado en un ordenador portátil equipado con un procesador Intel i7-7700HQ, y una unidad de procesamiento gráfico Nvidia GeForce GTX 1060. Para los experimentos de validación realizados en el mundo real, el procesamiento computacional es realizado *on-board*, empleando una Raspberry Pi 3 Model B.

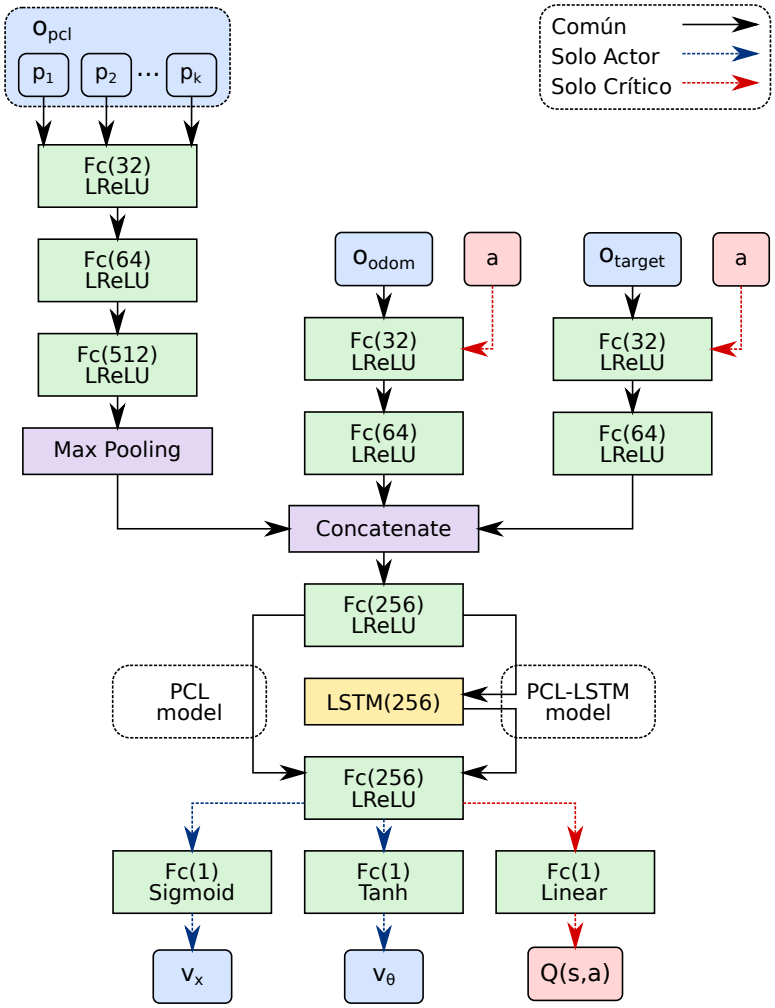
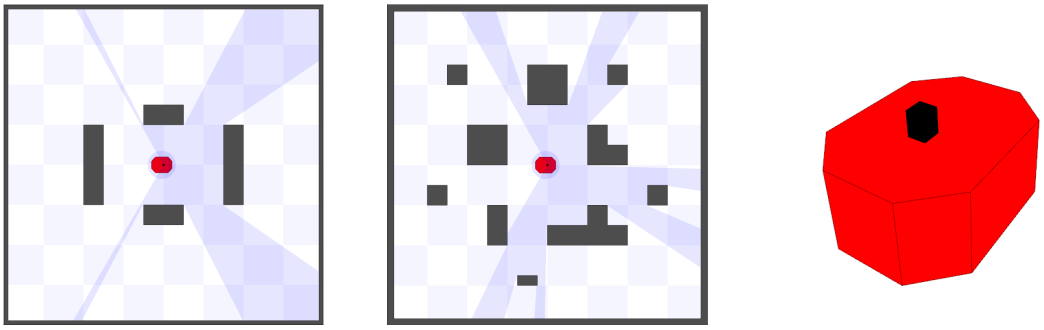


Figura 4.1: Parametrización multimodal para el actor y el crítico. Las capas son descritas siguiendo la notación "Tipo (parámetros) Función de activación", o simplemente "Tipo", en el caso de capas no paramétricas. "Fc" significa "*Fully connected*" y su parámetro representa el número de unidades ocultas asociado a la capa (esto también aplica para la capa LSTM). Finalmente, "LReLU" representa al término "Leaky ReLU".

4.4.1. Entrenamiento y evaluación en simulaciones

El entrenamiento de las políticas es conducido en los escenarios simulados propuestos en [94], y mostrados en la Figura 4.2. Por simplicidad, se hará referencia a estos escenarios como SW1 (Figura 4.2a) y SW2 (Figura 4.2b), respectivamente. Tanto SW1 como SW2 corresponden a espacios cerrados de 8 metros de largo por 8 metros de ancho, con obstáculos estáticos en su interior.

Para evaluar la efectividad del sistema propuesto, este es comparado con formulaciones alternativas, en las cuales varía tanto la definición de las observaciones del agente, como la parametrización de su política.



(a) Stage World 1 (SW1) **(b)** Stage World 2 (SW2) **(c)** Robot simulado

Figura 4.2: (a-b) Escenarios simulados en Stage para el entrenamiento de las políticas de planificación local. (c) Robot diferencial simulado. La geometría del robot simulado es consistente con la huella (*footprint*) del robot real. Las regiones sombreadas en azul en cada escenario representan el área cubierta por el sensor de rango montado en el robot. Adaptado de [26].

Formulaciones alternativas consideradas

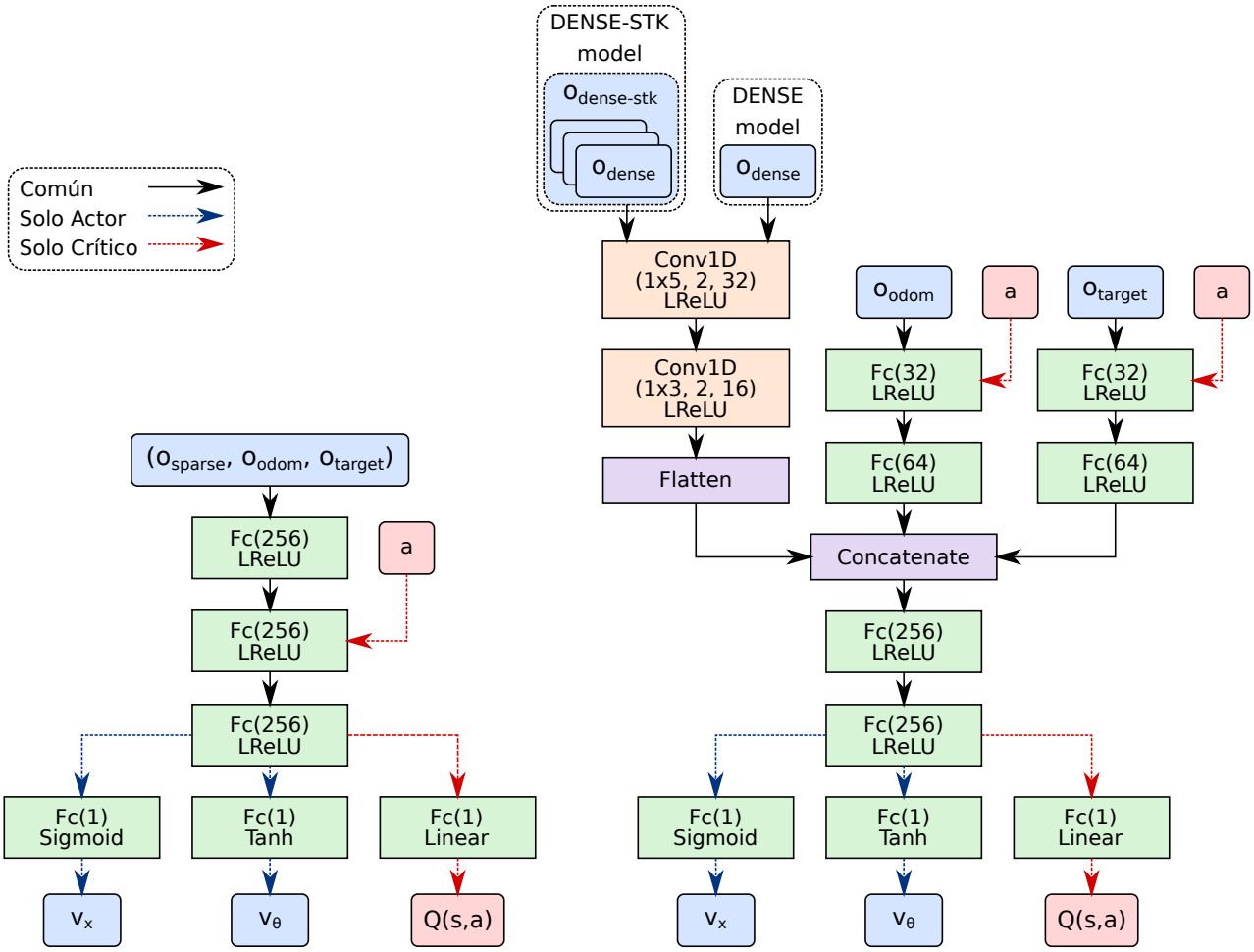
Como se señaló en la Sección 4.3.2, la representación de mediciones de rango como vectores ha sido explorada previamente al abordar el problema de navegación robótica con aprendizaje de máquinas. Con el fin de comparar el enfoque propuesto con esta clase de formulaciones, diferentes codificaciones para las observaciones del agente, así como parametrizaciones para su política, son estudiadas. Estas formulaciones se caracterizan por emplear vectores de tamaño fijo para la representación de las observaciones del agente, y extractores de características convencionales para procesarlas.

Las referencias de comparación consideradas se describen a continuación:

- SPARSE: Esta formulación sigue la mayoría de las decisiones de diseño propuestas en [23]: las observaciones del agente son construidas a partir de la concatenación y normalización de un vector de 10 mediciones de rango, o^t_{sparse}, la estimación de su velocidad instantánea, o^t_{odom}, y el objetivo de navegación en coordenadas polares, o^t_{target}, tomando como referencia al sistema de coordenadas local del robot. La parametrización empleada para el actor y el crítico también es muy similar a la propuesta en [23], salvo que un menor número de parámetros es empleado (256 unidades ocultas por capa *fully connected*, en lugar de 512).
- DENSE: En este caso, la componente de las observaciones que codifica las lecturas de rango es un vector 128-dimensional, o^t_{dense}. Al igual que en SPARSE, tanto la estimación de la velocidad instantánea del robot, o^t_{odom}, como el objetivo local de navegación en coordenadas polares, o^t_{target}, forman parte de las observaciones del agente. Debido a la gran diferencia entre la dimensión de o^t_{dense} al ser comparada con las dimensiones de o^t_{odom} y o^t_{target}, cada una de estas componentes es normalizada y entregada como entrada a las parametrizaciones del actor y el crítico siguiendo un enfoque multimodal. Adicionalmente, la componente o^t_{dense} es procesada utilizando capas convolucionales para manejar su alta dimensionalidad, técnica que ha sido empleada en trabajos previos (e.g. [94, 96, 97, 100]).
- DENSE-STK: Esta formulación sigue exactamente las mismas decisiones de diseño que DENSE, en este caso, no obstante, la codificación de las mediciones de rango correspon-

de a un *stack* de las mediciones instantáneas obtenidas en los instantes de tiempo t, $t-1$ y $t-2$, siendo cada una de estas mediciones un vector 128-dimensional. De este modo, la componente de las observaciones del agente que codifica esta información se define como $o^t_{\text{dense-stk}} = (o^{t-2}_{\text{dense}}, o^{t-1}_{\text{dense}}, o^t_{\text{dense}})$.

La Figura 4.3 muestra los diagramas asociados a las parametrizaciones de las formulaciones SPARSE (Figura 4.3a), y DENSE y DENSE-STK (Figura 4.3b). La notación empleada en estos diagramas sigue las convenciones utilizadas en las Figuras 3.5 y 4.1.



(a) SPARSE

(b) DENSE y DENSE-STK

Figura 4.3: Parametrizaciones para las formulaciones (a) SPARSE, y (b) DENSE y DENSE-STK.

Adicionalmente, se incluyen formulaciones alternativas que directamente emplean las parametrizaciones propuestas en otros trabajos de planificación local basada en aprendizaje reforzado.

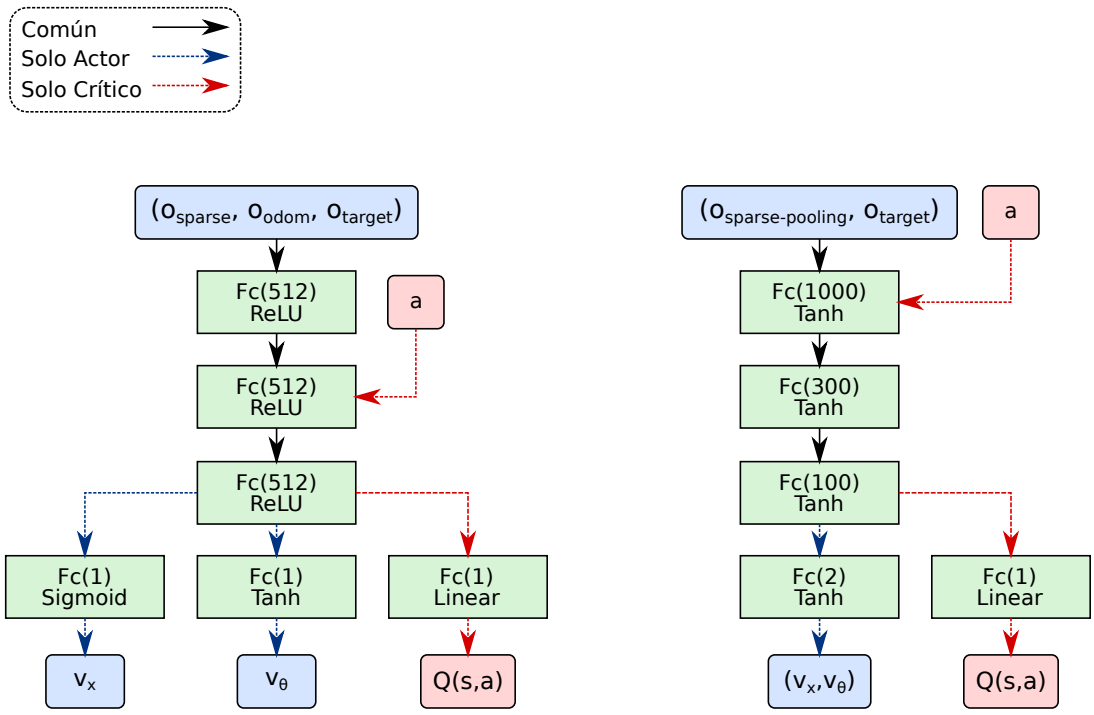
Estas formulaciones se describen a continuación:

- V2R: Las observaciones del agente y la parametrización empleada para el actor y el crítico, siguen las decisiones de diseño propuestas en [23]. En consecuencia, las observaciones y parametrizaciones para las redes neuronales son definidas como en SPARSE, salvo que, en este caso, la capa de entrada y las capas ocultas poseen un mayor número de parámetros (512

unidades ocultas por capa, en lugar de 256), y sus funciones de activación son de tipo ReLU, en lugar de Leaky ReLU.

- ASL: Las observaciones del agente, y la parametrización de su política, se construyen en función de las propuestas presentadas en [24]. En este caso, la codificación de las mediciones de rango para las observaciones del agente se realiza mediante un proceso de submuestreo, donde varias mediciones son agregadas según rangos angulares mediante la operación de *min pooling* (como se discutió brevemente en la Sección 4.3.2). Con esto, la componente de las observaciones que captura la información entregada por el sensor de rango se denota por $o^t_{\text{sparse-pooling}}$, y se compone por 36 mediciones.

La Figura 4.4 muestra los diagramas de las parametrizaciones asociadas a las formulaciones V2R (Figura 4.4a) y ASL (Figura 4.4b). Al igual que la Figura 4.3, la notación empleada en estos diagramas sigue las convenciones indicadas en las Figuras 3.5 y 4.1.



(a) V2R **(b)** ASL

Figura 4.4: Parametrizaciones para las formulaciones de referencia adicionales: (a) V2R [23], y (b) ASL [24].

Entrenamiento y evaluación

Las formulaciones de comparación SPARSE, DENSE y DENSE-STK, junto a las formulaciones propuestas en este trabajo, PCL y PCL-LSTM, son entrenadas en los escenarios mostrados en la Figura 4.2 (SW1 y SW2). Por otra parte, las formulaciones V2R y ASL, e instancias independientes de la formulación propuesta, PCL-LSTM, son entrenadas en "SW2r". El ambiente SW2r es casi idéntico a SW2, salvo que la función de recompensa empleada es modificada para que sea similar a

las que son utilizadas en [23] y [24]. En concreto, en SW2r la función de recompensa tiene la misma forma que la Ecuación (4.1), pero es redefinida de modo tal que $r^t_{\text{navigation}} = -C(\rho^t_{\text{target}} - \rho^{t-1}_{\text{target}})$, $r^t_{\text{success}} = 10$, y $r^t_{\text{collision}} = -0.4$, con $C = 2$. Los parámetros que definen a esta recompensa son tomados de [24], pero modificados ligeramente para abordar las diferencias existentes entre la dinámica de los agentes empleados en [24], y los empleados en este trabajo.

Si bien las observaciones del agente y la parametrización asociada a su política varía en cada formulación, en todos los casos las acciones son definidas de acuerdo a lo expuesto en la Sección 4.3.1. Adicionalmente, y para realizar una comparación justa, las políticas asociadas a cada formulación son entrenadas empleando los parámetros reportados en la Tabla 4.1. Los parámetros no reportados para DDPG, son directamente tomados de [34]. Para todos los agentes entrenados, además, el ruido de exploración es modulado por un factor que decae linealmente, de 1.0 a 0.05, en $5{\cdot}10^4$ pasos de tiempo. Finalmente, en el caso del agente PCL-LSTM, las experiencias son muestreadas como trazas de tamaño 16, y las actualizaciones sobre los pesos del actor y el crítico son omitidas por los primeros 4 elementos de dichas trazas. Para todos los otros agentes, las experiencias son muestreadas de manera uniforme.

	Parámetro	**Valor**
DDPG	Tasa de aprendizaje del actor	0.0001
	Tasa de aprendizaje del crítico	0.001
	Factor de suavizado λ	0.001
	Factor de descuento γ	0.99
	Tamaño máximo del *Experience Replay*	100000
	Tamaño del *mini batch*	256
Control	Frecuencia de control (Hz)	5.0
	$[v_x^{\min}, v_x^{\max}]$ (m/s)	[0.0, 0.5]
	$[v_\theta^{\min}, v_\theta^{\max}]$ (rad/s)	[-1.0, 1.0]

Tabla 4.1: Parámetros para el entrenamiento de las políticas de planificación local. Datos tomados de [26].

Para todos los experimentos, y al igual que en el caso de estudio de evasión de colisiones presentado en el Capítulo 3, el entrenamiento es conducido siguiendo una formulación episódica, donde cada episodio tiene una duración máxima de 500 interacciones agente-ambiente, y puede acabar anticipadamente si el robot colisiona o logra llegar al objetivo de navegación. En cada episodio, el agente parte con una pose fija en el origen del escenario de entrenamiento correspondiente (como se ilustra en las Figuras 4.2a y 4.2b), y un objetivo de navegación válido es generado de forma aleatoria.

Durante el entrenamiento, las mediciones de rango y las estimaciones de velocidad recibidas por el agente son ideales, y ninguna clase de perturbación artificial es incorporada a estas. Para obtener las coordenadas locales del objetivo de navegación en cada instante de tiempo, un sistema de localización que provee información "*ground-truth*" es utilizado.

Para todas las formulaciones estudiadas, el entrenamiento es conducido por 10^5 pasos de tiempo. Puesto que el algoritmo DDPG es utilizado, existe una correspondencia directa entre el número de pasos de tiempo en los que el agente interactúa con el ambiente, y el número de pasos de entrenamiento (i.e. el número de actualizaciones sobre los parámetros del actor y el crítico).

Cada $5{\cdot}10^3$ pasos de entrenamiento, los agentes son evaluados por 50 episodios (por lo que el ruido de exploración es deshabilitado durante este proceso). Para medir la evolución del desempeño de las políticas entrenadas, tres métricas son empleadas:

- Tasa de éxito: Corresponde a la razón entre el número de episodios en los que el agente logra llegar al objetivo de navegación, y el número de episodios en los que el agente es evaluado (en este caso, 50).
- Retorno promedio: Se calcula como el promedio de los retornos no descontados obtenidos por el agente en los episodios de evaluación.
- Interacciones agente-ambiente promedio: Corresponde al promedio de pasos de tiempo en que el agente interactúa con el ambiente antes de llegar a un estado terminal (por lograr llegar al objetivo de navegación, colisionar, o porque el límite de pasos de tiempo es alcanzado).

Para obtener resultados significativos, cada formulación estudiada es entrenada cinco veces de forma independiente. Cada proceso de entrenamiento (sin considerar la evaluación) toma entre 3 y 4 horas para los agentes SPARSE, DENSE, V2R y ASL, y entre 5 y 6 horas para los agentes DENSE-STK, PCL y PCL-LSTM. Las Figuras 4.5 y 4.6 muestran los resultados de evaluación obtenidos para todas las políticas entrenadas (promedio y desviación estándar considerando las cinco instancias entrenadas por cada formulación).

La Figura 4.5 muestra los resultados de evaluación asociados a los agentes SPARSE, DENSE, DENSE-STK, PCL y PCL-LSTM entrenados en SW1 (Figura 4.5a) y SW2 (Figura 4.5b), respectivamente. La Figura 4.6, por otra parte, muestra los resultados de evaluación asociados a los agentes que fueron entrenados en SW2r: V2R, ASL e instancias independientes de PCL-LSTM.

Los resultados obtenidos muestran que casi todos los agentes entrenados en SW1 y SW2 logran aprender políticas de planificación local que alcanzan buenos desempeños. Cuando el entrenamiento es conducido en SW1, se observa que las métricas de desempeño presentan una menor variabilidad entre agentes diferentes que cuando el entrenamiento es conducido en SW2. Este es un resultado esperable, pues SW2 posee una mayor complejidad que SW1.

Es posible apreciar, además, que algunos de las agentes entrenados presentan *peaks* en la métrica de interacciones agente-ambiente promedio. Los agentes que muestran esta tendencia logran adquirir rápidamente comportamientos de evasión de colisiones, con lo que son capaces de sobrevivir durante episodios completos, pero sin llegar al objetivo de navegación (los episodios acaban en *time-out* tras 500 pasos de tiempo).

4.4.2. Validación en simulaciones

Con el fin de evaluar la capacidad de generalización de los agentes entrenados, estos son desplegados en dos nuevos escenarios, esta vez simulados en Gazebo. Estos escenarios se muestran en la Figura 4.7, y se hará referencia a ellos como GW1 (Figura 4.7a) y GW2 (Figura 4.7b), respectivamente. Tanto GW1 como GW2 corresponden a espacios cerrados con paredes internas. Mientras que GW1 mide 10 metros de largo por 10 metros de ancho, GW2 mide 20 metros de largo por 20 metros de ancho.

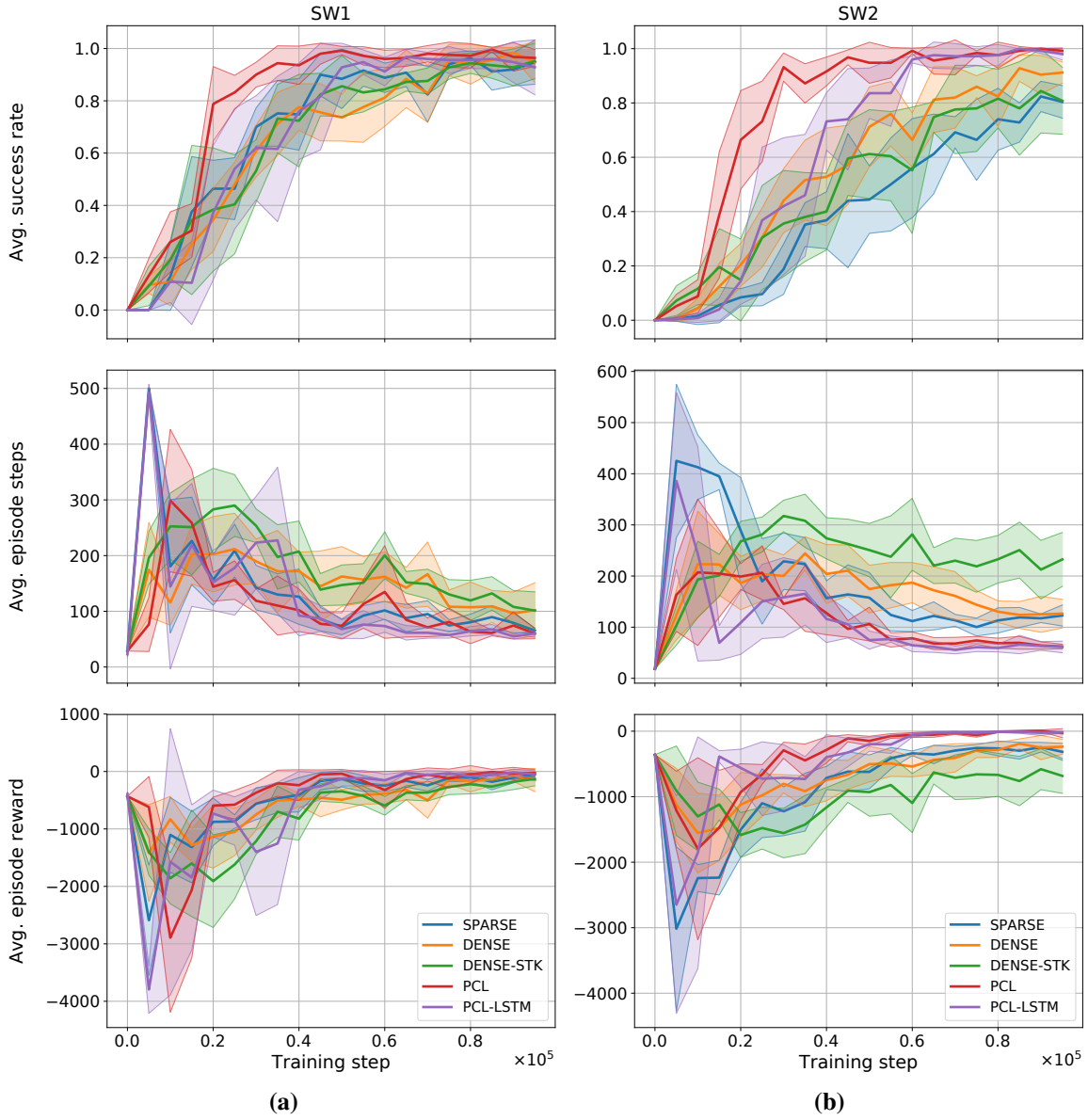


Figura 4.5: Evolución de la tasa de éxito promedio, número promedio de interacciones angente-ambiente, y retorno promedio durante el proceso de entrenamiento de los agentes SPARSE, DENSE, DENSE-STK, PCL y PCL-LSTM. (a) Muestra los resultados obtenidos al entrenar a los agentes en SW1, mientras que (b) muestra los resultados obtenidos al entrenarlos en SW2. Adaptado de [26].

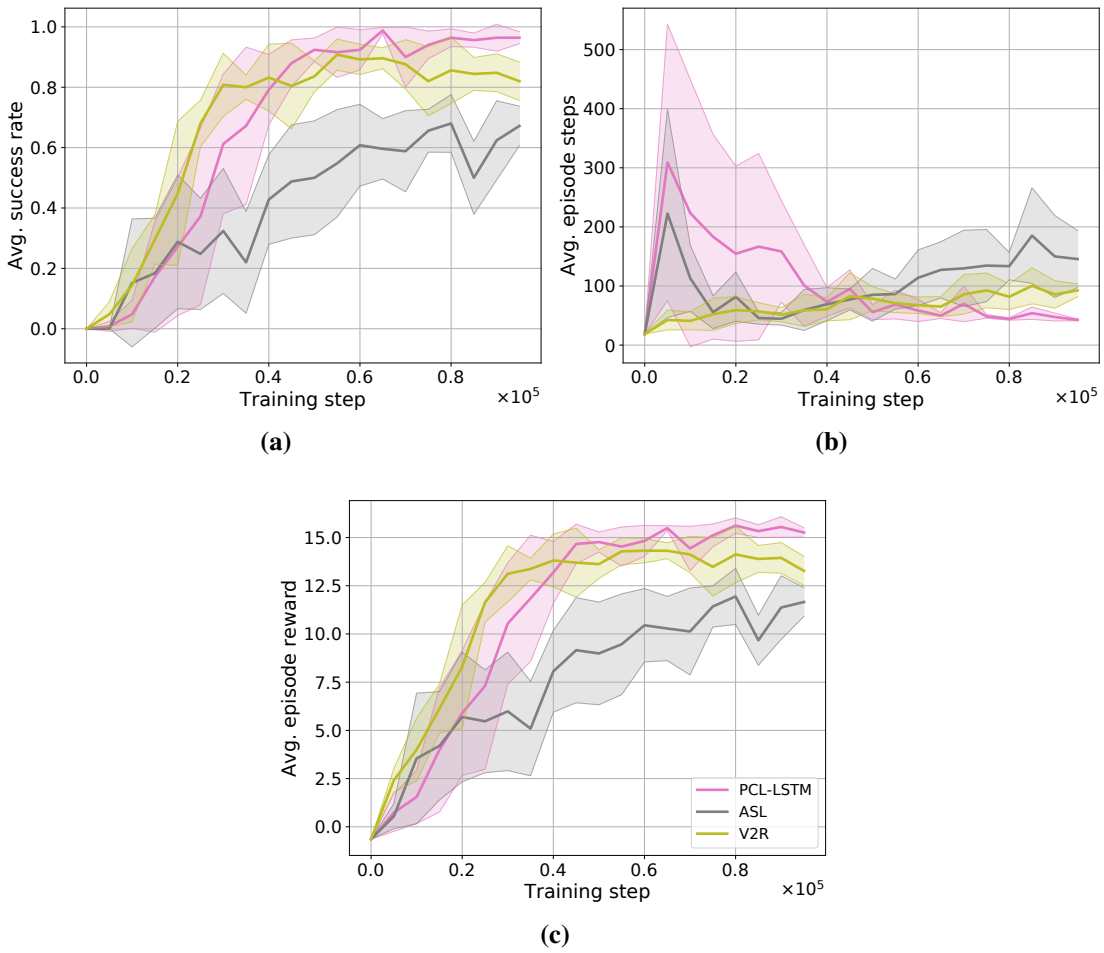


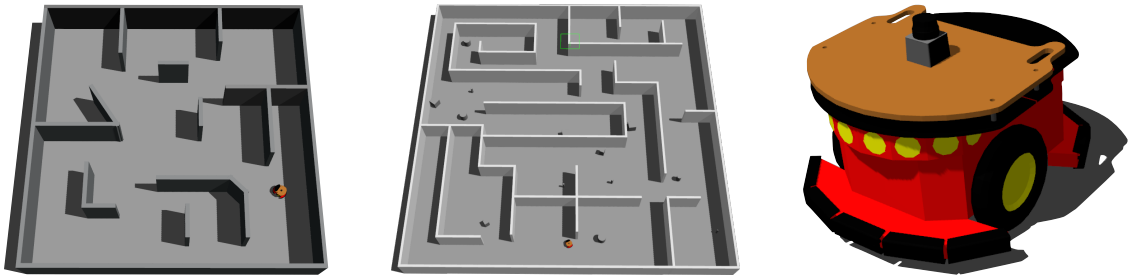
Figura 4.6: Evolución del desempeño de los agentes entrenados en SW2r: V2R, ASL y PCL-LSTM. (a) Tasa de éxito promedio, (b) número promedio de interacciones agente-ambiente, y (c) retorno promedio. Adaptado de [26].

Comparación entre las formulaciones alternativas consideradas

En primera instancia, todos los agentes entrenados son evaluados en GW1 durante 500 episodios. En cada episodio, los agentes deben lograr alcanzar objetivos de navegación generados aleatoriamente, partiendo desde el origen del escenario. Al igual que durante el proceso de entrenamiento, los agentes reciben las coordenadas locales del objetivo de navegación empleando un sistema de localización que provee información "*ground-truth*". Los episodios de evaluación terminan si el agente logra llegar al objetivo de navegación, colisiona, o se supera el límite de 500 interacciones agente-ambiente.

Para medir el desempeño de las políticas evaluadas, cuatro métricas de desempeño son utilizadas:

- Tasa de éxito (*Success Rate* [SR]): Es definida de forma equivalente a la tasa de éxito empleada para la evaluación de las políticas durante el entrenamiento. Corresponde a la razón entre el número de episodios en los que el agente logra llegar al objetivo de navegación, y el



(a) Gazebo World 1 (GW1) **(b)** Gazebo World 2 (GW2) **(c)** Robot simulado

Figura 4.7: (a-b) Escenarios simulados en Gazebo para la validación de las políticas de planificación local. (c) Pioneer 3-DX simulado. Adaptado de [26].

número total de episodios en que es evaluado (en este caso, 500).

- Tasa de *time-out* (*Time-out Rate* [TR]): Corresponde a la razón entre el número de episodios en los que el agente supera el límite de 500 interacciones agente-ambiente, y el número total de episodios de evaluación.
- Tasa de colisión (*Collision Rate* [CR]): Se define como la razón entre el número de episodios que termina prematuramente debido a una colisión del agente, y el número total de episodios de evaluación.
- Interacciones agente-ambiente promedio (*Avg. Steps*): Al igual que en el caso de la evaluación durante entrenamiento, se define como el promedio de pasos de tiempo en los que el agente interactúa con el ambiente antes de llegar a un estado terminal.

Los resultados obtenidos para esta primera evaluación son mostrados en la Tabla 4.2. El identificador ubicado en el subíndice del nombre asociado a cada agente, corresponde al escenario en el que dicho agente fue entrenado (SW1, SW2 o SW2r).

Agente	SR	TR	CR	Avg. Steps
$\text{SPARSE}_{\text{SW1}}$	0.74±0.14	0.02±0.01	0.25±0.14	87.7± 13.0
$\text{DENSE}_{\text{SW1}}$	0.64±0.11	0.02±0.02	0.34±0.11	123.8± 48.2
$\text{DENSE-STK}_{\text{SW1}}$	0.64±0.08	0.08±0.06	0.28±0.12	136.1± 39.6
PCL_{SW1}	0.85±0.02	0.06±0.05	0.09±0.05	95.8± 20.2
$\text{PCL-LSTM}_{\text{SW1}}$	0.88±0.03	0.08±0.02	0.04±0.02	100.0± 5.7
$\text{SPARSE}_{\text{SW2}}$	0.74±0.07	0.05±0.03	0.22±0.08	119.6± 15.9
$\text{DENSE}_{\text{SW2}}$	0.61±0.18	0.12±0.20	0.27±0.12	187.1±104.6
$\text{DENSE-STK}_{\text{SW2}}$	0.46±0.12	0.24±0.09	0.31±0.15	275.0± 51.1
PCL_{SW2}	0.86±0.04	0.06±0.03	0.08±0.06	95.7± 14.8
$\text{PCL-LSTM}_{\text{SW2}}$	0.88±0.02	0.09±0.02	0.02±0.02	103.4± 11.8
V2R_{SW2r}	0.59±0.11	0.24±0.08	0.17±0.09	186.5± 41.2
ASL_{SW2r}	0.36±0.12	0.12±0.12	0.52±0.23	151.8± 50.0
$\text{PCL-LSTM}_{\text{SW2r}}$	0.88±0.05	0.05±0.03	0.07±0.05	73.7± 11.4

Tabla 4.2: Resultados de evaluación obtenidos en GW1 para los agentes estudiados (promedio y desviación estándar considerando las cinco instancias entrenadas por cada agente). Datos tomados de [26].

Los resultados obtenidos muestran que los agentes PCL y PCL-LSTM presentan un mejor desempeño general que cualquier otra formulación alternativa considerada. Más aun, el desempeño de estos agentes en GW1 es muy similar, sin importar el escenario en el que fueron entrenados.

Al comparar a los agentes PCL y PCL-LSTM entrenados en SW1 y SW2, se observa que los agentes PCL-LSTM muestran un desempeño superior. Lo anterior se atribuye a la inclusión de capas LSTM a la parametrización del actor y el crítico de PCL-LSTM, considerando que este es el único punto que diferencia a esta formulación de PCL.

Los resultados también indican que de los agentes entrenados en SW2r, los agentes PCL-LSTM logran sobrepasar por un amplio margen de desempeño a los agentes V2R y ASL. Notablemente, se aprecia que los agentes PCL-LSTM alcanzan un SR prácticamente equivalente sin importar variaciones en la complejidad del ambiente de entrenamiento (SW1 versus SW2), o variaciones en la función de recompensa utilizada (SW1 y SW2 versus SW2r).

Al analizar los resultados obtenidos para los agentes SPARSE, DENSE y DENSE-STK, es posible observar que los agentes SPARSE superan en desempeño a los agentes DENSE y DENSE-STK. Al igual que PCL y PCL-LSTM, el desempeño de los agentes SPARSE en términos de SR no se ve particularmente alterado al variar el escenario de entrenamiento. Lo anterior podría deberse a que, al usar los agentes SPARSE observaciones de baja dimensionalidad (o_{sparse}), el proceso de aprendizaje es simplificado.

Si bien existen ligeras diferencias en desempeño para todos los agentes de la misma variante al ser entrenados en SW1 o SW2, el cambio más notable es observado para los agentes DENSE-STK, donde aquellos entrenados en SW1 superan en desempeño por un margen amplio a aquellos entrenados en SW2. Una posible explicación para justificar este resultado, es que la generación de *stacks* de observaciones para este agente, hace que sea propenso a sufrir un sobreajuste al ambiente en que es entrenado: los *stacks* de observaciones experimentados por un agente entrenado en un ambiente muy complejo, difícilmente serán observados nuevamente en un ambiente diferente.

En términos generales, los experimentos conducidos en GW1 demuestran empíricamente la importancia del diseño de las observaciones y parametrizaciones asociadas a un planificador local entrenado mediante aprendizaje reforzado, y muestran el gran impacto que estas decisiones de diseño tienen en el desempeño del mismo.

Evaluación de PCL-LSTM como planificador local en un sistema de navegación

Solo los agentes que alcanzaron el mejor desempeño promedio en el experimento anterior (los agentes PCL-LSTM$_{\text{SW2}}$) son sometidos a una nueva evaluación, esta vez conducida en GW2 (Figura 4.7b).

El escenario GW2 destaca por su complejidad: en él, la capacidad de planificación a largo plazo se vuelve necesaria para poder alcanzar los objetivos de navegación. Con el fin de abordar este requerimiento, en este experimento los agentes son guiados a los objetivos de navegación mediante una secuencia de *waypoints*. Estos *waypoints* son generados a partir de un muestreo sobre el plan

generado por el planificador global A* (*A-star*)[2]. Los *waypoints* son seleccionados en función de dos criterios: si la curvatura del plan global supera un cierto umbral, o si ningún otro *waypoint* ha sido seleccionado tras una cierta distancia fija, medida en función de la curva asociada al plan global. Ambos criterios son ajustados dependiendo de la granularidad del plan, que para este caso depende de la resolución del mapa métrico empleado por A*. En término prácticos, los criterios de selección se ajustan para que la distancia máxima entre *waypoints* sucesivos sea similar a la distancia máxima que recorrían las políticas PCL-LSTM$_{\text{SW2}}$ al ser evaluadas en simulaciones.

Adicionalmente, para verificar que los agentes entrenados pueden ser integrados a un sistema de navegación completo en el mundo real, en este experimento se emplea el sistema de localización AMCL (*Adaptive Monte Carlo Localization*)[3]. Finalmente, para tener una referencia con la cual contrastar el desempeño de los agentes entrenados, estos son comparados con el sistema de navegación `move_base`[4], el cual también hace uso de AMCL y A*.

Al igual que en el experimento realizado en GW1, los agentes son evaluados durante 500 episodios: en cada episodio deben alcanzar objetivos de navegación generados aleatoriamente, partiendo desde el origen del escenario. Dos variantes de GW2 son consideradas: una variante estática, a la que se hará referencia como GW2st, y una variante "aleatorizada", a la que se hará referencia como GW2rnd. En GW2st, los únicos obstáculos que los agentes deben soslayar para llegar a los objetivos de navegación, corresponden a las paredes fijas del escenario. En GW2rnd, en contraste, diferentes objetos son dispuestos en posiciones aleatorias al comienzo de cada episodio. Para ambas variantes, tanto AMCL como A* tienen acceso al mapa métrico que describe el posicionamiento de las paredes fijas de GW2.

Los resultados obtenidos para este experimento de evaluación se muestran en la Tabla 4.3. En este caso, el identificador ubicado en el subíndice del nombre asociado a cada agente corresponde al escenario en que es evaluado (GW2st o GW2rnd). Las mismas métricas de desempeño empleadas en el experimento de evaluación conducido en GW1 son utilizadas para este experimento.

Agente	SR	TR	CR	Avg. Steps
PCL-LSTM$_{\text{GW2st}}$	0.90±0.09	0.07±0.10	0.03±0.04	295.1±60.8
`move_base`$_{\text{GW2st}}$	0.85±0.05	0.13±0.07	0.02±0.02	287.2±56.3
PCL-LSTM$_{\text{GW2rnd}}$	0.67±0.09	0.17±0.11	0.16±0.04	313.0±42.6
`move_base`$_{\text{GW2rnd}}$	0.42±0.10	0.21±0.12	0.37±0.15	192.3±65.2

Tabla 4.3: Resultados de evaluación obtenidos en GW2 para PCL-LSTM y `move_base` (promedio y desviación estándar considerando cinco instancias independientes por agente). Datos tomados de [26].

Los resultados obtenidos, primeramente, muestran que los agentes PCL-LSTM pueden operar de forma apropiada al ser integrados a un sistema de navegación compuesto por diferentes módulos (en este caso, un módulo para localización, y un módulo para planificación global).

Adicionalmente, y considerando que por las características de GW2 el planificador A* siempre puede entregar un plan global válido, y que en general no existen problemas de localización durante

[2] `http://wiki.ros.org/action/fullsearch/global_planner`
[3] `http://wiki.ros.org/amcl`
[4] `http://wiki.ros.org/move_base`

los episodios de evaluación, se observa que el desempeño del sistema de navegación construido utilizando a los agentes PCL-LSTM supera el desempeño de `move_base`. En particular, se observa que en GW2st, el sistema `move_base` presenta un TR superior que PCL-LSTM, debido a que a menudo los agentes que emplean `move_base` en este escenario se estancan en pasajes estrechos.

Las diferencias de desempeño en GW2rnd son mucho más notorias. En primer lugar, ambos sistemas ven su desempeño disminuido al operar en GW2rnd debido a que este escenario presenta una mayor complejidad que GW2st, al poseer obstáculos que cambian de pose en cada episodio. Aún con esto, es posible apreciar que el desempeño de PCL-LSTM sigue siendo superior que el de `move_base`, resultado que se atribuye a su naturaleza reactiva: al emplearlo como planificador local, es capaz de seguir trayectorias libres de colisiones entre *waypoints*, aún cuando estas se alejan notoriamente del plan global.

Evaluación de PCL-LSTM frente a perturbaciones extremas en sus observaciones

Finalmente, con el fin de evaluar la respuesta de los agentes PCL-LSTM frente a perturbaciones extremas sobre sus observaciones, un tercer experimento de evaluación es conducido en GW1. En este experimento se estudian las variaciones de desempeño de los agentes al ser sometidos a (i) la restricción de su FoV, (ii) la limitación del número máximo de elementos que constituyen a la nube de puntos observada ($|o_{\text{pcl}}|_{\max}$), y (iii) la limitación del FoV de los agentes, pero compensada a través de la integración de las nubes de puntos observadas en el tiempo (AGGR). Esta integración de mediciones es realizada de forma equivalente a como se describe la creación de "mapas locales" en el caso de estudio de evasión de colisiones para el robot Pepper [25] (ver Sección 3.3.3).

Los resultados obtenidos para este experimento se muestran en la Tabla 4.4, donde las mismas métricas de desempeño utilizadas para los otros experimentos de evaluación en simulaciones son reportadas.

Variante			SR	TR	CR	Avg. Steps
FoV	$\|o_{\text{pcl}}\|_{\max}$	AGGR				
180°	128	✗	0.87±0.02	0.09±0.03	0.03±0.02	105.5±13.4
120°	128	✗	0.83±0.04	0.11±0.03	0.06±0.04	116.1±14.4
90°	128	✗	0.76±0.10	0.15±0.09	0.09±0.05	135.8±39.4
240°	64	✗	0.88±0.02	0.09±0.03	0.03±0.03	104.0±13.6
240°	32	✗	0.87±0.02	0.10±0.03	0.03±0.03	106.1±12.7
240°	16	✗	0.85±0.04	0.10±0.03	0.05±0.05	107.6±14.9
180°	128	✓	0.87±0.02	0.07±0.04	0.06±0.05	96.7±16.0
120°	128	✓	0.86±0.04	0.07±0.02	0.07±0.05	102.0±14.2
90°	128	✓	0.83±0.08	0.10±0.03	0.08±0.06	119.4±24.3

Tabla 4.4: Resultados de evaluación obtenidos en GW1 para los agentes PCL-LSTM al ser sus observaciones sometidas a diferentes perturbaciones (promedio y desviación estándar considerando las cinco instancias independientes entrenadas). Datos tomados de [26].

Los resultados muestran que la restricción el campo visual de los agentes PCL-LSTM, en todos los casos, hace que su desempeño se vea disminuido. Este es un resultado esperable, considerando que los agentes son entrenados empleando un sensor simulado que posee un FoV de 240°. Es posible notar, no obstante, que al incluir la integración de mediciones instantáneas en el tiempo (AGGR), la disminución de desempeño de los agentes al restringir su FoV es considerablemente menor.

También se observa que la disminución del número de puntos que componen a o^t_{pcl} solo disminuye ligeramente el desempeño de los agentes, incluso cuando esta disminución es extrema (por ejemplo, cuando $|o_{\text{pcl}}|_{\text{max}}$ es limitado a 16).

Los resultados obtenidos a través de este experimento refuerzan las hipótesis referentes a la robustez y flexibilidad del enfoque propuesto. Al ser estos agentes entrenados empleando entradas de tamaño variable, adquieren la capacidad de aprender representaciones intermedias que permiten emplear sensores con diferentes FoV y resoluciones al momento del despliegue del sistema. Más aún, estos cambios no implican un deterioro excesivo en el desempeño de los agentes, e incluso es posible suplirlos mediante técnicas de fusión sensorial, como se ha demostrado para el caso en el que se realiza la integración de mediciones en el tiempo.

4.4.3. Validación en el mundo real

Con el objetivo de evaluar la aplicabilidad del enfoque propuesto en el mundo real, PCL-LSTM es integrado al sistema de localización AMCL y al planificador global A*, y desplegado en ambientes de interior reales. Al igual que en los experimentos de evaluación conducidos en GW2, en este caso el agente PCL-LSTM es guiado a los objetivos de navegación a través de *waypoints* generados muestreando el plan global entregado por A*.

Si bien la percepción 2D con la que cuentan los robots simulados es suficiente para que estos se desenvuelvan adecuadamente en sus ambientes, la complejidad del mundo real hace que esta restricción limite fuertemente las capacidades de los agentes entrenados al ser desplegados en una plataforma real. Para aliviar este problema, la cámara RGB-D montada en el robot Pionner 3-DX utilizado, es empleada para proyectar las mediciones de rango entregadas por su sensor de profundidad a un arreglo de tamaño fijo, siguiendo un procedimiento como el descrito en [100]. Las mediciones obtenidas por la cámara RGB-D, son combinadas con aquellas que provienen del LiDAR antes de generar la nube de puntos que sirve como componente de las observaciones del agente entrenado. Esto permite al robot poder percibir objetos (dentro del FoV de la cámara) cuya geometría de colisión no puede ser capturada de forma correcta utilizando solo percepción 2D, por ejemplo, sillas y mesas.

El sistema de navegación es desplegado *on-board*, esto es, todo el procesamiento computacional requerido por el sistema de localización (AMCL), de planificación global (A*), adquisición y procesamiento de información, y la realización de inferencias empleando la política entrenada, es realizado en una Raspberry Pi 3 Model B, montada en el robot real. A pesar de esta restricción, el tiempo de inferencia promedio del actor, medido en el caso en que la nube de puntos o_{pcl} constantemente posee 128 elementos, es de aproximadamente 75 ms. El sistema, en su conjunto, es capaz de generar acciones de control consistentemente a una frecuencia de 5 Hz, que es la frecuencia de

control empleada durante los procesos de entrenamiento, evaluación, y validación conducidos en simulaciones.

Dos ambientes son considerados para realizar la validación en el mundo real: un ambiente especialmente construido para asemejarse a un espacio de interior, con paredes, puertas y obstáculos estáticos, y un corredor largo, con salas a los que el robot tiene acceso. En este último caso, ocasionalmente humanos actúan como obstáculos dinámicos. Cuando el robot es desplegado en ambos ambientes, una secuencia de objetivos de navegación son solicitados de manera secuencial. Ejemplos de las trayectorias recorridas por el robot son mostradas en la Figura 4.8. La ejecución de algunas de estas trayectorias puede ser visualizada en `https://youtu.be/AzvRJyN6rwQ`.

Para obtener una medida cuantitativa del desempeño del agente PCL-LSTM en el mundo real, tres experimentos son conducidos. Estos experimentos buscan evaluar el desempeño del agente al ser expuesto a situaciones particularmente difíciles: evasión de obstáculos cuya geometría de colisión solo es caracterizable con percepción 3D (evasión de sillas), control de alta granularidad (pasar a través de puertas) y planificación de corto plazo (atravesar pasajes bloqueados).

Para cada experimento, 30 episodios son realizados, y las métricas de desempeño SR, TR y CR, utilizadas en los experimentos de validación en simulaciones (ver Sección 4.4.2), son registradas. Para evaluar a la política entrenada de forma aislada, en estos experimentos solo se hace uso de AMCL como sistema de localización, y el plan global generado por A* es ignorado. Los resultados obtenidos son mostrados en la Tabla 4.5.

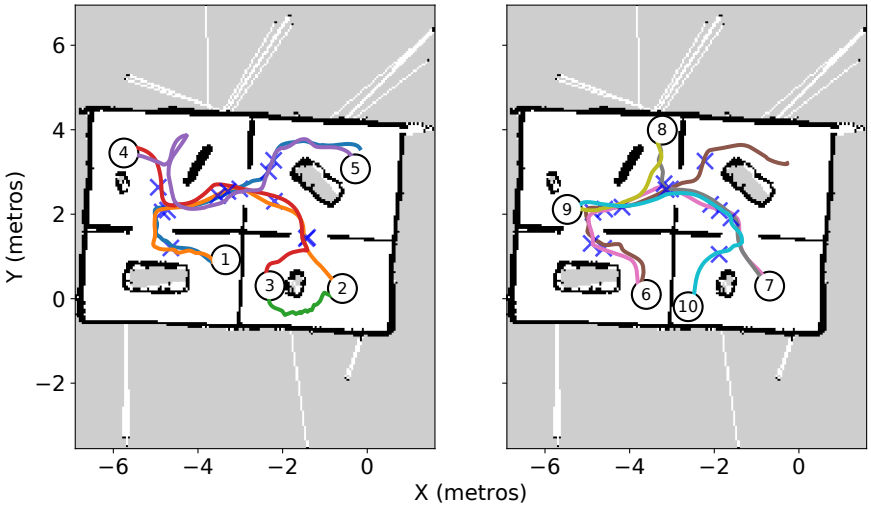
Experimento	SR	TR	CR
Evasión de sillas	0.93	0.00	0.07
Pasar a través de puertas	0.93	0.00	0.07
Atravesar pasajes bloqueados	0.87	0.03	0.10

Tabla 4.5: Resultados obtenidos para los experimentos de evaluación cuantitativa realizados en el mundo real. Datos tomados de [26].

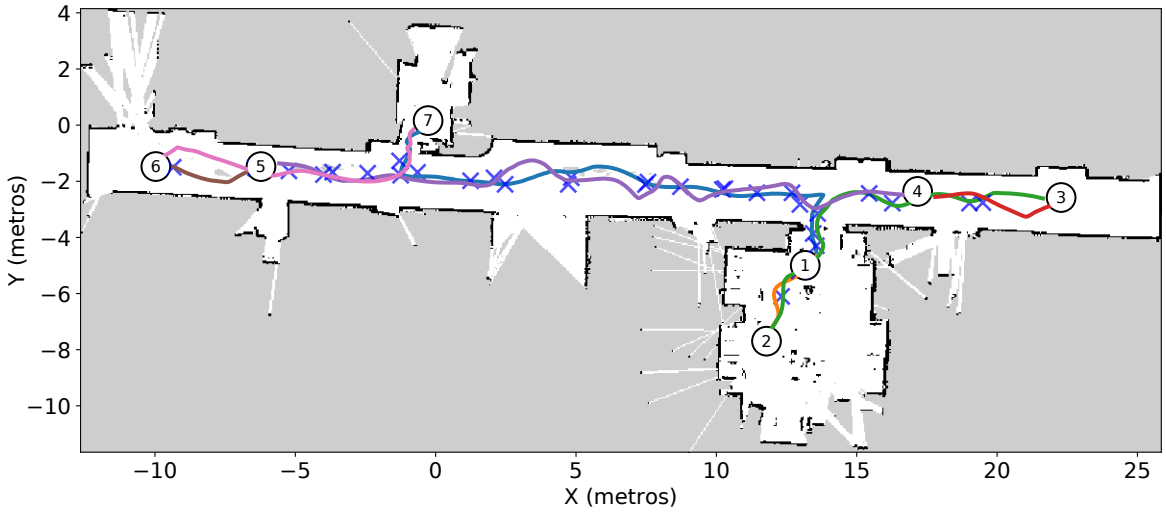
Los resultados obtenidos demuestran que el enfoque propuesto es capaz de desempeñarse de forma apropiada en el mundo real. En este sentido, se verifica experimentalmente que el planificador local desarrollado es integrable a un sistema de navegación completo, y es capaz de ser ejecutado en tiempo real en una plataforma computacionalmente limitada.

Más aún, el desempeño alcanzado es aceptable incluso considerando la presencia de retardos en la actuación del robot, mediciones ruidosas debido al uso de sensores reales, la presencia de obstáculos dinámicos, y errores de localización.

Adicionalmente, se muestra también que es posible dotar de una percepción 3D limitada al agente en tiempo de despliegue, mediante un simple procedimiento de fusión sensorial, sin que esto último sea considerado durante el entrenamiento del agente en simulaciones.



(a)



(b)

Figura 4.8: Ejemplos de las trayectorias ejecutadas por el agente PCL-LSTM al ser desplegado en (a) un ambiente de interior artificial, y (b) un corredor largo, con dos salas a las que el robot tiene acceso. En ambos casos, los números representan los objetivos de navegación secuenciales solicitados al robot, las líneas de colores a las trayectorias seguidas por el robot, y las cruces azules a los *waypoints* muestreados a partir del pan global generado por A*. Las trayectorias ejecutadas en el ambiente de interior artificial, han sido divididas en dos sub-figuras con el fin de facilitar su exposición. Adaptado de [26].

4.5. Discusión

En este capítulo se presentó una propuesta para la obtención de una política de planificación local mediante el uso de aprendizaje reforzado. Esta política es parametrizada mediante una red neuronal artificial, y permite mapear nubes de puntos 2D que codifican mediciones de rango, estimaciones de velocidad basadas en odometría y objetivos locales de navegación en coordenadas polares, a comandos de velocidad para un robot diferencial.

A partir de la realización de experimentos en simulaciones, fue posible verificar que el uso de nubes de puntos 2D para la codificación de mediciones de rango trae consigo importantes beneficios. En primer lugar, los agentes entrenados considerando esta representación lograron alcanzar desempeños finales superiores que formulaciones alternativas al ser desplegados en ambientes desconocidos. Adicionalmente, se comprobó que estos agentes sufren un detrimento muy leve en su desempeño al ser sus observaciones perturbadas de forma extrema.

El sistema de planificación local desarrollado fue desplegado en el mundo real, empleando al robot Pioneer 3-DX como plataforma física. Los experimentos conducidos en ambientes reales permitieron probar que el sistema es integrable a un *pipeline* de navegación completo, y que presenta un desempeño adecuado. Más aún, se pudo comprobar que el uso de técnicas de pre-procesamiento y/o fusión sensorial es factible al momento de despliegue, y permite, por ejemplo, aumentar las capacidades perceptuales del agente.

Una de las limitaciones del sistema propuesto, es que no cuenta con la capacidad de planificación a largo plazo. Esto hace que se haga necesario el uso de información global del ambiente para permitir que el agente llegue a objetivos de navegación en escenarios laberínticos. Cuando esta información no está disponible, es usual que la política de planificación local muestre comportamientos oscilatorios. Una posible dirección de investigación que podría ser explorada para solucionar este problema, consiste en el uso de memorias externas (e.g. [101]).

Capítulo 5

Conclusión y trabajo futuro

En esta tesis se han abordado dos problemas asociados a navegación robótica: la evasión de colisiones, y la planificación local. Empleando aprendizaje reforzado, dos sistemas independientes han sido propuestos con el fin de dar solución a estos problemas, siendo ambos sistemas desplegables en el mundo real.

El primer sistema desarrollado permitió dotar al robot Pepper con la capacidad de evadir colisiones. Extensivas evaluaciones en simulaciones fueron conducidas para verificar la estabilidad del enfoque propuesto, y las ventajas prácticas referentes al uso de nubes de puntos para la representación de mediciones de rango. En particular, se pudo comprobar que esta clase de representaciones permite mejorar las capacidades perceptuales del agente cuando las mediciones instantáneas utilizadas para su construcción son integradas en el tiempo. Más aún, al emplear una parametrización multimodal para las políticas entrenadas, se comprobó que el uso de nubes de puntos es compatible con otra clase de representaciones para las observaciones del agente (e.g. imágenes y/o vectores). Por otro lado, los experimentos de validación conducidos empleando al robot físico, permitieron verificar que el sistema desarrollado es desplegable en el mundo real, presentando un desempeño aceptable en ambientes considerablemente diferentes a los que fueron empleados durante el proceso de entrenamiento.

El segundo sistema desarrollado abordó el problema de planificación local. En este caso, parte de las observaciones del agente fueron representadas como nubes de puntos 2D, y su utilización fue comparada con representaciones alternativas, propuestas en trabajos anteriores. Mediante la realización de experimentos en simulaciones, se pudo comprobar que la utilización de nubes de puntos 2D como representación para mediciones de rango, permite la obtención de políticas robustas a perturbaciones extremas sobre las observaciones del agente. Más aún, se comprobó que el desempeño final de las políticas entrenadas al ser desplegadas en ambientes desconocidos (diferentes al de entrenamiento) resulta ser superior al de las formulaciones alternativas que fueron estudiadas. Mediante la realización de experimentos en el mundo real, finalmente, se verifica que la propuesta es integrable a un sistema de navegación completo, y que sus propiedades permiten usar estrategias de pre-procesamiento y/o fusión de datos sensoriales para mejorar las capacidades perceptuales del agente en tiempo de despliegue.

Lo anterior permite concluir que los principales objetivos de este trabajo fueron cumplidos sa-

tisfactoriamente. Más aún, y en el contexto del aprendizaje reforzado de políticas para navegación, se logró comprobar experimentalmente que el uso de nubes de puntos 2D para la representación de las observaciones de los agentes trae importantes beneficios, lo que respalda la hipótesis de este trabajo.

Si bien los casos de estudio desarrollados mostraron resultados satisfactorios, existen diferentes direcciones de investigación que pueden ser exploradas para extender el trabajo realizado.

En primer lugar, una característica de los sistemas desarrollados es que no cuentan con la capacidad de planificación a largo plazo. Esto no es necesariamente problemático en el caso del sistema de evasión de colisiones, no obstante, impone limitaciones en el desempeño del sistema de planificación local. Este último requiere contar con información global del ambiente para poder llegar a objetivos de navegación que podrían ser difíciles de alcanzar al considerar escenarios laberínticos. Si bien el uso de capas LSTM en la parametrización de la política es beneficioso, no permite capturar dependencias temporales de largo plazo para lidiar con esta clase de situaciones, donde es probable que el agente muestre comportamientos oscilatorios. Considerando este antecedente, si el enfoque es puesto en el desarrollo de un sistema de planificación que pueda desempeñarse de forma efectiva en ambientes laberínticos, la utilización de técnicas tales como el uso de memorias externas (e.g. [101, 102, 103]) podría ser beneficioso.

En segundo lugar, en esta tesis las nubes de puntos empleadas para representar las observaciones de los agentes entrenados fueron construidas usando puntos 2D. Aunque esta decisión de diseño es adecuada (y natural) al emplear sensores LiDAR 2D como los que fueron usados en los casos de estudio presentados, no lo es necesariamente al contar con sensores que permitirían otorgar al agente con percepción 3D. Considerando esto, y que el extractor de características que fue empleado en este trabajo es adecuado para el procesamiento de puntos n-dimensionales, una dirección interesante de estudio sería la construcción de nubes de puntos con puntos de más alta dimensionalidad, con el objetivo de proveer al agente con una representación mas rica de su entorno.

Finalmente, en este trabajo solo se abordaron dos problemas específicos ligados a navegación robótica, pero la propuesta general es extrapolable a problemas diferentes. En este sentido, una dirección interesante de investigación podría ser caracterizar mediante nubes de puntos las observaciones de agentes en problemas diferentes a los estudiados, como por ejemplo, otros sub-problemas asociados a navegación, como exploración autónoma, mapeo o localización, o incluso a problemas diferentes donde caracterizar al ambiente mediante el uso de nubes de puntos de largo variable sea conveniente.

Bibliografía

[1] Nils J Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *Proceedings of the First International Joint Conference on Artificial Intelligence*, pages 509–520, 1969.

[2] Georges Giralt, Ralph Sobek, and Raja Chatila. A multi-level planning and navigation system for a mobile robot: a first approach to hilare. In *Proceedings of the 6th international joint conference on Artificial intelligence-Volume 1*, pages 335–337, 1979.

[3] Charles Thorpe, Martial H Hebert, Takeo Kanade, and Steven A Shafer. Vision and navigation for the carnegie-mellon navlab. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):362–373, 1988.

[4] Georges Giralt, Raja Chatila, and Marc Vaisset. An integrated navigation and motion control system for autonomous multisensory mobile robots. In *Autonomous robot vehicles*, pages 420–443. Springer, 1990.

[5] Guilherme N DeSouza and Avinash C Kak. Vision for mobile robot navigation: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 24(2):237–267, 2002.

[6] Alonzo Kelly, Anthony Stentz, Omead Amidi, Mike Bode, David Bradley, Antonio Diaz-Calderon, Mike Happold, Herman Herman, Robert Mandelbaum, Tom Pilarski, et al. Toward reliable off road autonomous vehicles operating in challenging environments. *The International Journal of Robotics Research*, 25(5-6):449–483, 2006.

[7] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[9] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[10] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[11] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

[12] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[13] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[14] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[15] OpenAI: Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[16] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[17] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[18] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters*, 5(4):6670–6677, 2020.

[19] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[20] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1–8. IEEE, 2018.

[21] Ole-Magnus Pedersen, Ekrem Misimi, and François Chaumette. Grasping unknown objects by coupling deep reinforcement learning, generative adversarial networks, and visual servoing. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5655–5662, 2020.

[22] Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. Towards monocular vision based obstacle avoidance through deep reinforcement learning. *arXiv preprint arXiv:1706.09829*, 2017.

[23] Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE, 2017.

[24] Mark Pfeiffer, Samarth Shukla, Matteo Turchetta, Cesar Cadena, Andreas Krause, Roland Siegwart, and Juan Nieto. Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations. *IEEE Robotics and Automation Letters*, 3(4):4423–4430, 2018.

[25] Francisco Leiva, Kenzo Lobos-Tsunekawa, and Javier Ruiz-del Solar. Collision avoidance for indoor service robots through multimodal deep reinforcement learning. In Stephan Chalup, Tim Niemueller, Jackrit Suthakorn, and Mary-Anne Williams, editors, *RoboCup 2019: Robot World Cup XXIII*, pages 140–153, Cham, 2019. Springer, Springer International Publishing.

[26] Francisco Leiva and Javier Ruiz-del Solar. Robust rl-based map-less local planning: Using 2d point clouds as observations. *IEEE Robotics and Automation Letters*, 5(4):5787–5794, 2020.

[27] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[28] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018.

[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[30] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[31] 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.

[32] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[33] Yuxi Li. Deep reinforcement learning. *arXiv preprint arXiv:1810.06339*, 2018.

[34] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[35] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[36] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[37] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[38] Anil K Jain. *Fundamentals of digital image processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[39] Hanie Sedghi, Vineet Gupta, and Philip M Long. The singular values of convolutional layers. *arXiv preprint arXiv:1805.10408*, 2018.

[40] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[41] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[42] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

[43] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[44] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[45] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[46] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, 2000.

[47] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30:5099–5108, 2017.

[48] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9397–9406, 2018.

[49] Zhiyuan Zhang, Binh-Son Hua, and Sai-Kit Yeung. Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1607–1616, 2019.

[50] John N Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

[51] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[52] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395, 2014.

[53] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870, 2018.

[54] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596, 2018.

[55] Amit Kumar Pandey and Rodolphe Gelin. A mass-produced sociable humanoid robot: pepper: the first machine of its kind. *IEEE Robotics & Automation Magazine*, 25(3):40–48, 2018.

[56] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[57] Johann Borenstein and Yoram Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on systems, Man, and Cybernetics*, 19(5):1179–1187, 1989.

[58] Johann Borenstein, Yoram Koren, et al. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 7(3):278–288, 1991.

[59] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[60] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[61] Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.

[62] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[63] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[64] Lei Tai, Shaohua Li, and Ming 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. IEEE, 2016.

[65] Shichao Yang, Sandeep Konam, Chen Ma, Stephanie Rosenthal, Manuela Veloso, and Sebastian Scherer. Obstacle avoidance through deep networks based intermediate perception. *arXiv preprint arXiv:1704.08759*, 2017.

[66] Tony J Prescott and John EW Mayhew. Obstacle avoidance through reinforcement learning. In *Advances in neural information processing systems*, pages 523–530, 1992.

[67] Jeff Michels, Ashutosh Saxena, and Andrew Y Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 593–600, 2005.

[68] Bing-Qiang Huang, Guang-Yi Cao, and Min Guo. Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance. In *2005 International Conference on Machine Learning and Cybernetics*, volume 1, pages 85–89. IEEE, 2005.

[69] Lei Tai and Ming Liu. Towards cognitive exploration through deep reinforcement learning for mobile robots. *arXiv preprint arXiv:1610.01733*, 2016.

[70] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[71] Keyu Wu, Mahdi Abolfazli Esfahani, Shenghai Yuan, and Han Wang. Learn to steer through deep reinforcement learning. *Sensors*, 18(11):3650, 2018.

[72] Keyu Wu, Han Wang, Mahdi Abolfazli Esfahani, and Shenghai Yuan. Bnd*-ddqn: Learn to steer autonomously through deep reinforcement learning. *IEEE Transactions on Cognitive and Developmental Systems*, 2019.

[73] Abhik Singla, Sindhu Padakandla, and Shalabh Bhatnagar. Memory-based deep reinforcement learning for obstacle avoidance in uav with limited environment knowledge. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[74] Guan-Horng Liu, Avinash Siravuru, Sai Prabhakar, Manuela Veloso, and George Kantor. Learning end-to-end multimodal sensor policies for autonomous navigation. In *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 249–261. PMLR, 13–15 Nov 2017.

[75] Naman Patel, Prashanth Krishnamurthy, Yi Fang, and Farshad Khorrami. Reducing operator workload for indoor navigation of autonomous robots via multimodal sensor fusion. In *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 253–254, 2017.

[76] Naman Patel, Anna Choromanska, Prashanth Krishnamurthy, and Farshad Khorrami. A deep learning gated architecture for ugv navigation robust to sensor failures. *Robotics and Autonomous Systems*, 116:80–97, 2019.

[77] Kenzo Lobos-Tsunekawa, Francisco Leiva, and Javier Ruiz-del Solar. Visual navigation for biped humanoid robots using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 3(4):3247–3254, 2018.

[78] Luona Yang, Xiaodan Liang, Tairui Wang, and Eric Xing. Real-to-virtual domain unification for end-to-end autonomous driving. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 530–545, 2018.

[79] Chuong V Nguyen, Shahram Izadi, and David Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *2012 second international conference on 3D imaging, modeling, processing, visualization & transmission*, pages 524–530. IEEE, 2012.

[80] Sung-Yeol Kim, Manbae Kim, and Yo-Sung Ho. Depth image filter for mixed and noisy pixel removal in rgb-d camera systems. *IEEE Transactions on Consumer Electronics*, 59(3):681–689, 2013.

[81] Minora Asada. A report on robocup 2017 [competitions]. *IEEE Robotics & Automation Magazine*, 24(4):21–23, 2017.

[82] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[83] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, January 1972.

[84] Alexandru Telea. An image inpainting technique based on the fast marching method. *Journal of graphics tools*, 9(1):23–34, 2004.

[85] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[86] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[87] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[88] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[89] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 1527–1533. IEEE, 2017.

[90] Wei Gao, David Hsu, Wee Sun Lee, Shengmei Shen, and Karthikk Subramanian. Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation. In *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 185–194. PMLR, 13–15 Nov 2017.

[91] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *International Conference on Learning Representations*, 2017.

[92] 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 *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

[93] Carlos Sampedro, Hriday Bavle, Alejandro Rodriguez-Ramos, Paloma de la Puente, and Pascual Campoy. Laser-based reactive navigation for multirotor aerial robots using deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1024–1031. IEEE, 2018.

[94] Linhai Xie, Sen Wang, Stefano Rosa, Andrew Markham, and Niki Trigoni. Learning with training wheels: speeding up training with a simple controller for deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6276–6283. IEEE, 2018.

[95] Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5113–5120. IEEE, 2018.

[96] Jun Jin, Nhat M Nguyen, Nazmus Sakib, Daniel Graves, Hengshuai Yao, and Martin Jagersand. Mapless navigation among dynamics with social-safety-awareness: a reinforcement learning approach from 2d laser scans. *arXiv preprint arXiv:1911.03074*, 2019.

[97] Ayzaan Wahid, Alexander Toshev, Marek Fiser, and Tsang-Wei Edward Lee. Long range neural navigation policies for the real world. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 82–89. IEEE, 2019.

[98] Maximilian Hüttenrauch, Sosic Adrian, Gerhard Neumann, et al. Deep reinforcement learning for swarm systems. *Journal of Machine Learning Research*, 20(54):1–31, 2019.

[99] Richard Vaughan. Massively multi-robot simulation in stage. *Swarm intelligence*, 2(2-4):189–208, 2008.

[100] Jinyoung Choi, Kyungsik Park, Minsu Kim, and Sangok Seok. Deep reinforcement learning of navigation in a complex and crowded environment with a limited field of view. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5993–6000. IEEE, 2019.

[101] Jingwei Zhang, Lei Tai, Joschka Boedecker, Wolfram Burgard, and Ming Liu. Neural slam: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*, 2017.

[102] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In *International Conference on Learning Representations*, 2018.

[103] Arbaaz Khan, Clark Zhang, Nikolay Atanasov, Konstantinos Karydis, Vijay Kumar, and Daniel D. Lee. Memory augmented control networks. In *International Conference on Learning Representations*, 2018.