UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

DOCTORADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN
SEÑALES, IMÁGENES Y AMBIENTES VIRTUALES

NAVEGACIÓN VISUAL USANDO APRENDIZAJE REFORZADO E IMÁGENES RGB-D

TESIS

QUE PARA OPTAR POR EL GRADO DE:

DOCTOR EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:

CARLOS ADRIÁN SARMIENTO GUTIÉRREZ

TUTOR PRINCIPAL:
DR. JESÚS SAVAGE CARMONA
FACULTAD DE INGENIERÍA

CIUDAD UNIVERITARIA. CD. MX. MAYO, 2022

Agradecimientos

Este trabajo fue financiado por CONACYT, a través de la beca asignada por el Posgrado en Ciencia e Ingeniería de la Computación. IIMAS. UNAM.

También agradezco a DGAPA-UNAM por el financiamiento, a través del proyecto PAPIIT AG101721 *"Modelos computacionales para el comportamiento adaptable de robots de servicio y de humanos"*.

Agradezco sinceramente ...

... A mis padres Rosaura y Pablo Alejandro ... ¡Por todo!

... A mi hermano Christian y su esposa Jacqueline, por su apoyo y cariño.

... A Tabata Melisa ...
Por las vacaciones que nunca pude tomar... Los lugares que nunca visité...
La comida que nunca probé... Los paisajes que nunca contemplé...
La brisa que nunca respiré... Hasta que te conocí...

Por todo tu amor y esfuerzo...

... A mis compañeros de equipo y amigos en este viaje:
Edgar S., Jesus, Hugo, Edgar G., Vivian y Erik.

... A los camaradas y amigos de los inicios... Por el apoyo desde siempre:
Víctor, Fernando, César, Raymundo y Telésforo.

... Al Dr. Savage por su tiempo y consejos. Por su trabajo para formar en el área de robótica de servicio.

Resumen

En la actualidad, los robots y vehículos autónomos han tenido un gran desarrollo debido al auge en nuevas tecnologías como son las cámaras digitales. Algunas de estas cuentan con excelente resolución y velocidad de captura, además del avance en nuevos métodos de aprendizaje de máquina. Ahora resulta más fácil tener acceso a sistemas electrónicos de gran calidad para desarrollar aplicaciones de visión por computadora, junto con la gran innovación que se lleva a cabo desde hace dos décadas en sistemas de aprendizaje profundo. Esto brinda la capacidad para interpretar la información proveniente de los sensores generando una descripción útil del ambiente, solamente utilizando imágenes digitales. Uno de estos casos es el utilizar sistemas de visión basados en redes neuronales convolucionales aplicadas a distintos sistemas de visión artificial con el objetivo de filtrar, segmentar, describir y clasificar contenido visual de interés para los sistemas de percepción y navegación en robots móviles autónomos.

Un ejemplo de estos sistemas de visión aplicados a la navegación, es el uso del aprendizaje reforzado como mecanismo de inferencia para encontrar políticas de ejecución de acciones en un robot que requiere moverse en su entorno de trabajo. La utilización exclusivamente de imágenes como entradas a los sistemas, impone desafíos técnicos, como es la eficiencia en el uso de los datos. Una manera de solventar este problema es utilizar representaciones digitales del entorno, en conjunto con un motor de física.Lo que permite la interacción de un modelo digital de un robot y su ambiente de manera más realista. A estos sistemas se les conoce como *simuladores fotorrealistas*.

En este trabajo se describe un nuevo sistema de navegación visual utilizando imágenes digitales obtenidas por un simulador fotorrealista, aprendizaje reforzado y una memoria topológica visual basada en similitud estadística. La finalidad es dotar a un robot móvil, con procedimientos para ejecutar acciones de navegación utilizando solo imágenes digitales. Este sistema ha sido probado en un sistema de simulación, con cinemática y dinámica incorporadas con el fin de facilitar su adaptación a agentes reales.

Abstract

Nowadays, autonomous robots and vehicles have had a great development due to the increase in new technologies such as digital cameras. Some of these have excellent resolution and capture speed, as well as advances in new machine learning methods. It is now easier to have access to high-quality electronic systems to develop computer vision applications, along with the great innovation that has been carried out for two decades in deep learning systems. This provides the ability to interpret the information coming from the sensors showing a useful description of the environment, using only digital images. One of these cases is to use vision systems based on convolutional neural networks applied to different artificial vision systems with the aim of filtering, segmenting, describing and classifying visual content of interest for the perception and navigation systems in autonomous mobile robots.

An example of these vision systems applied to navigation is the use of reinforcement learning as an inference mechanism to find action execution policies in a robot that requires movement in its work environment. The exclusive use of images as inputs to the systems imposes technical challenges, such as the efficiency in the use of data. One way to get around this problem is to use digital representations of the environment, in conjunction with a physics engine. This allows the interaction of a digital model of a robot and its environment in a more realistic way. These systems are known as *photorealistic simulators*.

In this work, a new visual navigation system is described using digital images obtained by a photorealistic simulator, reinforcement learning and a visual topological memory based on in statistical similarity. The purpose is to provide a mobile robot with procedures to execute navigation actions using only digital images. This system has been tested in a simulation system, with kinematics and dynamics incorporated in order to facilitate its adaptation to real agents.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En la última década los robots han comenzado a ser aplicados fuera del ramo industrial en donde han sido extensamente utilizados para tareas altamente estructuradas. En la actualidad se pretende que desarrollen trabajos más comunes y menos específicos, como fue el ensamblaje y transporte de componentes. La expansión de su campo de aplicación a dado lugar a una nueva rama de la robótica llamada *robótica de servicio* [1]. Esto se debe a que las tareas más genéricas son por mucho más complicadas de resolver que las tareas repetitivas en la industria. La federación internacional de robótica plantea el concepto *robot de servicio* como:

"Un robot que opera de manera automática o semiautomática para realizar servicios útiles al bienestar de los humanos o a su equipamiento, excluyendo las operaciones de fabricación."

En la actualidad hay pocos ejemplos de robots de servicio completamente funcionales, pero existen lugares donde se realiza investigación alrededor del mundo dedicados a la robótica de servicio como son universidades en Estados Unidos, Europa, y algunos laboratorios especializados en la automatización de sistemas. En un futuro se espera que se desarrolle un mercado de consumo personal para este tipo de robots.

Los robot de servicio son todos aquellos diseñados con capacidades de convivir con personas y de ejecutar tareas comunes que realizan las mismas. Algunos tipos de robots de servicio son los robots domésticos o robots de vigilancia, robots de limpieza y ordenamiento de espacios, entre otros.

Un robot de servicio doméstico debe realizar sus tareas sin intervención de las personas excepto durante su programación [2]. Otras funciones requeridas en el comportamiento de un robot doméstico son la manipulación de objetos como platos, vasos y utensilios que se encuentran en el hogar. También el ordenar espacios y estantes. Dichos

robots aún no existen de manera comercial, aunque en algunos centros de investigación se está trabajando en su desarrollo como es el Laboratorio de Biorrobótica pertenciente al Posgrado de Ingeniería, en la UNAM.

1.1. Motivación

La presente investigación surge de la necesidad de crear robots de servicio más eficientes, que se asemejen en su comportamiento a un ser humano. Es decir, que sean capaces de tomar desiciones y de desplazarse en un entorno real con la información que reciben de su sistema de visión. Además, la constante mejora en sistemas de imágenes digitales y su bajo costo hace posible el llevar a cabo experimentos rápidamente y con calidad utilizando simulaciones de última generación.

Los robots de servicio en la actualidad son una área de desarrollo en continuo crecimiento, por lo que la mejora de los sistemas incorporados en estos robots, eventualmente permitirá su aplicación real en tareas cotidianas. La investigación en esta área de conocimiento resulta muy atractiva e incluso es aplicable a otras ramas de la ciencia fuera de la robótica de servicio.

En el libro [3] se define que la navegación está compuesta por dos procesos: la *búsqueda del camino* y la *locomoción*. La búsqueda de un camino resuelve problemas de orientación y toma de decisiones, se procura elegir la ruta más apropiada hacia un lugar con características bien definidas. La locomoción coordina los movimientos usando información del sistema sensorial y del sistema de locomoción.

En el diseño de robots móviles se han utilizado múltiples clases de sensores pero en particular se han desarrollado sistemas basados en visión artificial usando cámaras digitales de distintos tipos, como son a colores, en escala de grises, cámaras de visión nocturna entre otras, según la aplicación para la que se haya diseñado el robot.

Los sistemas de visión resultan adecuados ya que la percepción visual ofrece mucha información útil del entorno y aumenta el alcance de las aplicaciones. Un ejemplo es el que un robot puede ser capaz de navegar, detectar y manipular objetos, reconocer personas y lugares, con tan solo la información proveniente de su sistema de visión.

Se define el *problema de localización* como la estimación de la posición de un robot móvil, dicha posición puede ser relativa o local respecto de las mediciones. De manera global si se tiene un sistema de referencia como un mapa del entorno. O de manera estadística, local o gloabal, generando una probabilidad de estar en un lugar determinado. El problema de localización es la parte fundamental de los sistemas de navegación en un

robot autónomo. Debido a la naturaleza dinámica del ambiente estos sistemas deben ser robustos a perturbaciones, variaciones del entorno y oclusiones, además ser capaces de actualizar los registros de las locaciones que han visitado.

Los sistemas de localización pueden clasificarse como sistemas globales o sistemas locales.

- La *localización local* estima la posición del robot basada en las lecturas de los sensores y se hace una predicción en el momento. Se limita a rastrear la posición hasta el punto donde el robot comenzó a operar.
- La *localización global* es más complicada. Este tipo de localización aproxima la posición actual del robot respecto al entorno completo usando solo las observaciones actuales.

  Un problema fundamental de la localización global es la de inicializar los sistemas cuando se enciende al robot sin tener información de donde ha estado anteriormente, usando solo la información que recibe de sus sensores.

Por otra parte, el concepto de *navegación* se define como la planeación de las acciones para lograr llegar un lugar determinado en el ambiente. Para esto, es fundamental el realizar una primera estimación de la localización, establecer el destino, crear un plan de movimiento y ejecutarlo. Todo de manera autónoma y considerando cambios en el entorno. El problema de navegación es complicado de resolver ya que involucra muchos tipos de subsistemas como son sistemas sensoriales, reconocimiento de objetos, sistemas de control de movimiento, odometría, mapeo, sistemas de evasión de obstáculos, etc. Estos subsistemas deben estar integrados compartiendo la capacidad de cómputo y el sistema de distribución de energía del robot.

Los sistemas de navegación pueden clasificarse en dos grandes clases: sistemas de navegación en espacion interiores y sistemas para navegación en espacios exteriores. Cada uno a su vez, tiene subclases que se listan a continuación:

- Navegación en interiores.
  - Navegación basada en mapas. Son los sistemas de navegación con una descripción previa del entorno nombrada comúnmente como un mapa. Este mapa puede ser una representación geométrica como un conjunto de polígonos, o un grupo de celdas ocupadas y libres, o un mapa que representa la relación en el espacio de las características que se han observado, conocido como mapa topológico.

    - Navegación basada en construcción de mapas. En este tipo de navegación el robot es capaz de dezplazarse por el entorno mientras construye una representación del mismo. Una técnica popular es la *localización y mapeo simultáneos* (LMS).
    - Navegación sin mapas. En este tipo de navegación el robot no cuenta con un mapa, solo tiene disponible la información sensorial en el momento. Algunas técnicas son el flujo óptico [4] y la búsqueda de correspondencias basada en apariencia [5].

- Navegación en exteriores.

    - Navegación en ambientes estructurados. Esta es la navegación en ambientes abiertos pero con algún tipo de estructura como la pintura para indicar la separación de los carriles en las carreteras o la apariencia de un sendero en caminos de terracería. Esta técnica es popularmente conocida como *seguir el camino*.
    - Navegación en ambientes no estructurados. Estos sistemas de navegación son los más demandantes ya que no existe algún tipo de estructura o patrón constante que el robot pueda rastrear, por lo que debe explorar el entorno para construir un mapa con las mediciones de algún sistema de percepción y utilizar los datos que ha recolectado por sí mismo.

- Navegación visual.

    - Con las definiciones anteriores podemos definir la navegación visual. La *navegación visual* es la tarea de planeación de acciones para llegar a un objetivo o un lugar utilizando únicamente sensores de imágenes. Igualmente pueden aplicarse todas las subcategorias mencionadas extendiendo está definición para ambientes estructurados, no estructurados, interiores o exteriores.

En la actualidad, la mayoría de sistemas de navegación utilizan sistemas basados en la localización y mapeo simultáneo (LMS) de puntos característicos en imágenes digitales. Este problema se define en la ecuación 2.2, en el capítulo 2. De manera breve podemos mencionar que este tipo de planteamiento requiere de algoritmos complejos para el cálculo de rotaciones y traslaciones sobre los puntos detectados en distintos sistemas de coordenadas, con el objetivo de construir un mapa tridimensional del ambiente. Este problema puede abordarse utilizando aprendizaje reforzado y la disponibilidad de ambientes y agentes simulados, lo que resulta una ventaja en el desarrollo de nuevos algoritmos de navegación visual.

Además, el uso de sistemas entrada-salida reduce la complejidad de los sistemas en un robot, pero también puede aportar robustez a los sistemas ya utilizados incorporando la experiencia explorada durante el proceso de entrenamiento. Otra ventaja, es que las redes neuronales profundas pueden utilizar información de distintas fuentes y esta puede ser fusionada mediante concatenación de vectores, para su posterior uso como entradas. Este proceso es más difícil de llevar a cabo en sistemas LMS ya que el proceso de óptimización es meramente geométrico. Por lo que nuevas fuentes de información requieren otro tratamiento.

1.2. Planteamiento del problema

Este trabajo escrito expone la implementación de un sistema de navegación visual para el caso de uso de un robot virtual de servicio. Este sistema está basado en el uso de sensores de información visual, utilizando aprendizaje reforzado como mecanismo de inferencia. Este tipo de sistemas ha comenzado ha desarrollarse debido a la disponibilidad de los sensores de imágenes y con el propósito de utilizar herramientas emergentes como plataformas de desarrollo de videojuegos que se pueden aplicar al ciclo de desarrollo de aplicaciones en robots de servicio.

Para el caso de sistemas basados en aprendizaje reforzado, es una problemática en investigación el integrar información del ambiente de manera global o local para obtener un sistema de tipo *entrada-salida* en donde se logre encontrar una política de ejecución de acciones. Lo anterior, con la menor cantidad de datos de exploración o únicamente con imágenes como entrada al sistema.

Este problema se aborda utilizando una memoria auxiliar construida a partir de imágenes capturadas previamente en el ambiente. Después, suministrando esta información global al sistema y de manera permanente se pretende reducir la cantidad de imágenes requeridas para el entrenamiento. Además de mejorar la ejecución de acciones de movimiento.

1.3. Hipótesis

El problema de navegación utilizando imágenes da lugar a las siguientes hipótesis:

- La navegación en un robot móvil puede llevarse a cabo solamente utilizando técnicas de aprendizaje profundo para imágenes, aprendizaje por refuerzo e información

topológica por similitud visual. Dando lugar a diseñar un sistema con memoria auxiliar visual del ambiente y navegación.

- Se puede construir una memoria visual auxiliar usando dos métodos de clasificación probabilística para entrenar un sistema basado en aprendizaje por refuerzo con mejor desempeño. Los métodos son modelos ocultos de Markov (MOM) y hash sensible a la localidad (HSL), ya que ambos métodos establecen un modelo probabilístico del contenido en las imágenes agregando robustez al sistema.

1.4. Contribuciones y trabajos publicados

Las contribuciones de este trabajo son:

- Un método de clasificación de imágenes utilizando modelos ocultos de Markov.
- Aplicación de métodos probalísticos para la búsqueda de imágenes semejantes.
- Un método de navegación visual incorporando una memoría topológica visual del ambiente y aprendizaje reforzado.

Resultado directo de este trabajo de tesis son las siguientes publicaciones:

- *Comparison of Two Objects Classification Techniques using Hidden Markov Models and Convolutional Neural Networks.* Sarmiento Carlos, Savage Jesus. Informatics and Automation (SPIIRAS Proceedings 2020). Vol 19. No 6. Diciembre 2020.
  https://doi.org/10.15622/ia.2020.19.6.4
  (Indexado en Scopus).
- *Feature detection using Hidden Markov Models for 3D-visual recognition.* Carlos Sarmiento, J. Savage, et al. IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC2019).
  https://doi.org/10.1109/ICARSC.2019.8733651
  (Indexado en Scopus).

1.5. Estructura de la tesis

En el capítulo 2, se hace una revisión de qué es el problema de navegación visual y como lograr que un sistema robótico navegue a un objetivo en el ambiente. Se aborda

la problemática desde el punto de vista basado en visión por computadora, aprendizaje profundo y aprendizaje por refuerzo. En el capítulo 3, se hace un resumen de los conceptos fundamentales del aprendizaje por refuerzo y como es posible aplicar las teorías y algoritmos a la planeación de ejecución de acciones de movimiento.

Dentro del capítulo 4 y en el capítulo 5 se desarrollan los métodos utilizados en este trabajo, repectivamente son los modelos ocultos de Markov (MOM) y el hash sensible a la localidad (HSL). El capítulo 6 expone como se utilizan estos métodos para construir una memoria del ambiente.

El capítulo 7 describe como se utiliza aprendizaje reforzado y la memoria visual por apariencia para resolver el problema de navegación visual y objetivos visuales.

En el capítulo 8 se presenta la metodología y las herramientas de software usadas para desarrollar el sistema de navegación, además de los distintos resultados obtenidos de la propuesta.

Por último, en el capítulo 9 se exponen las conclusiones de la investigación y las posibles mejoras al sistema de navegación visual propuesto.

Capítulo 2

Antecedentes

En este capítulo se resume la literatura relacionada con la navegación visual aplicada a robots autónomos, se recopilan ejemplos de investigación realizada en sistemas robóticos simulados y con robots reales. Estos trabajos se han realizado utilizando como fuente principal de información las imágenes capturadas por una cámara digital, real o simulada.

Los avances relacionados con la navegación visual utilizando aprendizaje por refuerzo se describe en la sección 2.1. En la sección 2.2 se hace una comparación de la tarea de que un robot pueda navegar en un entorno usando únicamente imágenes y las dificultades de estos sistemas.

2.1. Sistemas basados en aprendizaje profundo

Los sistemas de navegación visual han cobrado relevancia debido al gran aumento en la producción de los sensores en cámaras digitales y de la necesidad de bajar los costos en sistemas de percepción. Por esto, es una tendencia la construcción de sistemas de visión por computadora cada vez más complejos, tanto en la academia como en el área industrial.

Los sistemas de navegación visual con mayor uso en sistemas robóticos autónomos, están basados en *localización y mapeo simultáneos* (LMS). A su vez, una variante que utiliza una cámara como sensor principal se denomina *LMS visual*. Con las imágenes capturadas de los sensores se crea una representación geométrica del ambiente [6]. En comparación con sensores del tipo luz láser, las imágenes retienen información semántica del ambiente que puede ser rastreada en el tiempo, con el propósito de construir, optimizar y localizarse en un mapa.

El problema principal que se plantea en sistemas de localización y mapeo simultáneos, es estimar la localización del sistema de referencia local de la cámara utilizando un modelo matemático que relaciona los valores de los píxeles con coordenadas en un sistema de referencia global. Esto se logra sabiendo los parámetros de la óptica del sensor para calcular en que coordenadas en la imagen digital serán proyectados puntos en el mundo real, lo cual requiere un proceso de calibración previo. Después, mediante el desplazamiento y rotación de la cámara es posible predecir el cambio de posición de los píxeles en una secuencia de imágenes, lo que da lugar a inferir la posición de dichos puntos en el sistema de referencia global. Este proceso se describe matemáticamente como un problema de minimización en la proyección de un conjunto de puntos de un sistema coordenado a otro [7], es decir:

$$\{\mathbf{R},\mathbf{t}\} = \arg\min_{\mathbf{R},\mathbf{t}} \sum_{i\in\chi} \rho\left(\| \mathbf{x}^i_{(\,\cdot\,)} - \pi_{(\,\cdot\,)}(\mathbf{R}\mathbf{X}^i + \mathbf{t}) \|^2_{\Sigma}\right) \tag{2.1}$$

Donde ρ es una función de costo como es la función de Huber [8], utilizada para eliminar valores atípicos. La matriz Σ es una matriz de covariancia asociada a la escala para detectar puntos característicos. Las matrices $\mathbf{R},\mathbf{X},\mathbf{t}$ son respectivamente la rotación, los puntos característicos ubicados en un sistema de referencia global, y el vector de traslación para encontrar los puntos $\mathbf{x}^i_{(\,\cdot\,)}$ en la imágen asociada. Finalmente, $\pi_{(\,\cdot\,)}$ es el modelo de la cámara que se utiliza para proyectar los puntos en el modelo espacial de la imagen digital.

Adicionalmente, para construir el mapa del entorno de manera simultánea se requiere la optimización constante de una plantilla inicial, por lo que se debe agregar este proceso de reestimación a todos los cálculos anteriores. Incluyendo las latencias de todos los subsistemas y su sensibilidad a estas.

El problema de construcción de un mapa de manera simultánea se define como sigue: sean P_L todos los puntos de las tomas K_L covisibles; sean K_F las tomas restantes y no en K_L, y todo el conjunto de punto detectados entre imágenes $\chi_k \in P_L$, siendo k una *toma clave* seleccionada por algún mecanismo de búsqueda para eliminar redundancia. Entonces tenemos:

$$\{\mathbf{X}^i,\mathbf{R}_l,\mathbf{t}_l \mid i \in P_L, l \in K_L\} = \arg\min_{\mathbf{X}^i,\mathbf{R}_l,\mathbf{t}_l} \sum_{k\in K_L\cup K_F} \sum_{j\in\chi_k} \rho(E_{k,j}) \tag{2.2}$$

Con $E_{k,j}$ definido como:

$$E_{k,j} = \| \mathbf{x}^{j}_{(\,.\,)} - \pi_{(\,.\,)}(\mathbf{R}\mathbf{X}^{j} + \mathbf{t}) \|^{2}_{\Sigma} \tag{2.3}$$

El problema en la ecuación 2.2 suele requerir implementaciones muy sofisticadas y que requieren muchos procesos de ejecución en procesadores con muy baja latencia. Sin embargo es la norma en sistemas de navegación, si se cuenta con buenos sensores y computadoras capaces.

En el trabajo de Klein y otros [9] se propone el rastreo en paralelo de puntos de interés en el ambiente para su mapeo, denomidado *rastreo y mapeo en paralelo* (RMP). El método RMP resuelve el problema planteado en la ecuación 2.2 pero con una mejora, dicha ecuación requiere de la captura y selección de las tomas, la extracción de puntos característicos, su comparación y adición al modelo geométrico para su posterior optimización, todo con el objetivo de disminuir el error de reproyección. RMP desacopla el proceso de obtención de características visuales y el proceso de comparación y estimación usando predicciones sobre los píxeles ya observados, esto utilizando procesos en paralelo en el procesador que ejecuta el sistema. La idea general es que un proceso principal calcula el error de la posición de la cámara según un sistema de referencia en movimiento, mientras que otro proceso construye un mapa en tres dimensiones con los puntos localizados. Estos procesos tienen a su vez subprocesos ejecutándose. Para poder lograr la convergencia se utilizan estimaciones locales más frecuentes y una estimación global en un periodo más grande, evitando la solución global de la ecuación 2.2 en cada ciclo.

Mur-Artal [10] propone una mejora a RMP, utilizando nuevas caracterísitcas visuales con invariancia a rotación y de fácil obtención, en lo que se conoce como el descriptor ORB (siglas del inglés *Oriented and fast robust local feature detector and Rotated Binary robust independent features*). El resultado final de este proceso se muestra en la figura 2.1.

Algunos métodos de LMS son capaces de procesar la textura en las imágenes de manera directa como describe el autor Engel en [11], utilizando la diferencia de valores entre imágenes de profundidad a gran escala con fines de navegación en ambientes exteriores. Este método incorpora mejoras en la velocidad de cálculo al evitar calcular descriptores en cada nueva imagen procesada, además de usar odometría visual.

Los sistemas de localización y mapeo basados en LMS, mayormente requieren un modelo del movimiento del sensor o de la óptica del mismo, para poder reconstruir la posición en un sistema de referencia para cada píxel. Este problema requiere modelos complejos y algunos supuestos sobre como es el ruido y la aplicación específica del sistema. Otra desventaja, es la de requerir extracción de características o algoritmos complejos, con procesamiento paralelo para obtener resultados prácticos. También es importante mencionar

Figura 2.1: Ejemplo de estimación de posición usando localización y mapeo simultáneos con imágenes. Los polígonos azules representan el sistema de referencia de la cámara respecto de un sistema global de coordenadas. Los puntos coloreados son píxeles reproyectados en el sistema global de coordenadas (imagen tomada de [10]).

que estos sistemas únicamente construyen la representación del ambiente y localizan la posición del sensor de donde se extrae la información. Para poder tener un sistema de navegación completo es necesario tener en cuenta la integración de la posición a sistemas de navegación, la planeación y su ejecución.

En la actualidad, hay una línea de investigación en constante avance para utilizar aprendizaje profundo para ejecutar tareas de extracción de características visuales o de navegación visual, incorporando la percepción, planeación y navegación en un solo sistema de aprendizaje. A continuación se exponen algunos trabajos basados en aprendizaje profundo para realizar la tarea de navegación visual.

Vlad Mnih y otros [12], en la empresa *DeepMind* en el año 2013 (propiedad de Google en el 2014) retoman investigaciones anteriores, ya establecidas desde la década de 1990 [13] en la aplicación de sistemas de aprendizaje de máquina, usando una señal de recompensa como condicionamiento de un programa a un objetivo. Además, con el aumento de la capacidad de cómputo y el surgimiento del mercado de tarjetas gráficas, viejos algoritmos de aprendizaje (principalmente basados en búsquedas dentro de tablas), ahora pueden implementarse usando redes neuronales. Mnih describe el mecanismo de entrenamiento de un agente diseñado para jugar un conjunto de viejos juegos de la consola *Atari*. La principal aportación fué el uso de una red convolucional profunda, entrenada usando un algoritmo desarrollado con base en la ecuación 3.14 y la ecuación 3.15. La figura 2.2 muestra la arquitectura utilizada.

La red neuronal convolucional se compone de varios niveles denominados *capas ocul-*

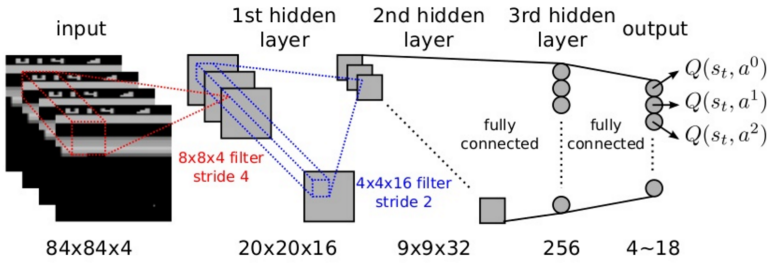


Figura 2.2: Arquitectura propuesta por Mnih para jugar videojuegos de la consola Atari (imagen tomada de [12]).

tas. Estas utilizan un conjunto de filtros y mediante su aplicación se detectan características visuales. La primera capa implementa 16 filtros de 8x8x4 con un salto de 4 píxeles por ventana. La segunda capa implementa 32 filtros de 4x4x16 y un salto de 2 píxeles, esta capa genera un vector de salida de 256 valores. Dichos valores se alimentan a una capa completamente conectada con 4 a 18 neuronas de salida según el videojuego a entrenar. Cada capa además ejecuta una función de activación. Las funciones de activación son de la forma rectificador lineal excepto en la capa de salida, la cual es una función de activación lineal. La función Q de la figura 2.2, asocia una acción predeterminada con el mayor valor de activación dado el estado actual s_t, representado por las imágenes en la entrada.

El sistema aprende a ejecutar acciones mapeando una representación de estados, y los estados se modelan como características visuales obtenidas por la red convolucional. Durante el primer millón de acciones, el agente utiliza una técnica de entrenamiento denominada *política voraz*.

El agente es condicionado obteniendo una recompensa positiva si ejecuta una acción que incrementa su marcador. Si no incrementa su marcador se le penaliza. Se hacen ejecuciones de entrenamiento que finalizan cuando el agente pierde. Después se reinicia el juego y se continúa con el entrenamiento. A esto se le conoce como entrenamiento por episodios.

La aportación de este método fue que se puede implementar un sistema *entrada-salida*, en donde las entradas son observaciones directas del ambiente y las salidas son valores que se asocian a la mejor acción a ejecutar, dado un objetivo.

Usando gradiente descendente, los bancos de filtros se entrenan de manera automática, al igual que la capa de neuronas que selecciona las acciones, maximizando el valor de la función Q. La fragilidad de este sistema es que viola la *propiedad de Markov de primer orden* ya que se usan tres imágenes anteriores a la actual, dicha propiedad solo permite

un estado anterior como memoria. Este problema es atenuado entrenando indirectamente las observaciones, mediante el uso de almacenamiento temporal en lo que se denomina *memoria de repetición* [14]. Esta memoria almacena 4-tuplas de imágenes, posteriormente cada tupla es elegida aleatoriamente para romper su correlación temporal. Otra desventaja, es que si hay ruido en las imágenes las acciones ejecutadas se vuelven de naturaleza aleatoria ya que no se tiene ningún mecanismo de regularización o de inmunidad al ruido. Dicho problema se resuelve usando una etapa de extracción de características más robusta al ruido, a cambios de iluminación y perspectiva, lo que implica entrenarla fuera de linea. Incluso existen métodos que proponen agregar perturbaciones a los coeficientes de la red lo que favorece la convergencia y robustez [15].

En el año 2019, Wang y otros [16] proponen usar aprendizaje por refuerzo para entrenar un agente que cumpla con la tarea de navegar a un objetivo visual, usando una representación de estados que se obtienen a través de tres mecanismos principales: el primero es usar una red convolucional denominada *ResNet-50* [17] para extraer características visuales; el segundo es suministrar una palabra que describa el objetivo que se desea alcanzar, como puede ser la palabra *televisor*, *cafetera*, *mesa*, etcétera. Y esa palabra es transformada en un vector mediante la biblioteca *fastText* [18]; el tercero es usar información semántica del ambiente. Esto mediante la construcción de un grafo que contiene entre sus nodos los objetos en un determinado entorno, y en las aristas la relación semántica entre los objetos, como puede ser *arriba*, *debajo*, *a la izquierda*, etc. Esto se hace usando las clases en el conjunto de datos *VisualGenome* [19]. La figura 2.3 muestra la arquitectura para dicho trabajo.

Este enfoque resulta novedoso ya que propone la fusión de información del ambiente, aprendizaje reforzado implementado de manera asíncrona y el uso de un simulador como mecanismo de entrenamiento, el cual fue desarrollado usando *Unity* (un entorno de desarrollo de videojuegos).

El simulador utilizado es nombrado *THOR* (siglas del inglés *The House Of inteRactions*) y es desarrollado por el Instituto de Allen para la inteligencia artificial, en Washington [20]. Dicho simulador es uno de los entornos actualmente mejor implementado para el entrenamiento de agentes basados en visión, ya que incorpora un motor de física, interacciones sobre los objetos y una apariencia fotorrealista, lo que facilita la adecuación de los modelos entrenados a una plataforma real. En este simulador se condiciona al agente con una recompensa de valor $r = 10$ si se llega al objetivo, o $r = -0.1$ si falla. Se usa entrenamiento por episodios, por lo que el simulador se reinicia cuando: el agente colisiona; se llega al objetivo en el ambiente; o se alcanza un número de iteraciones como condición de

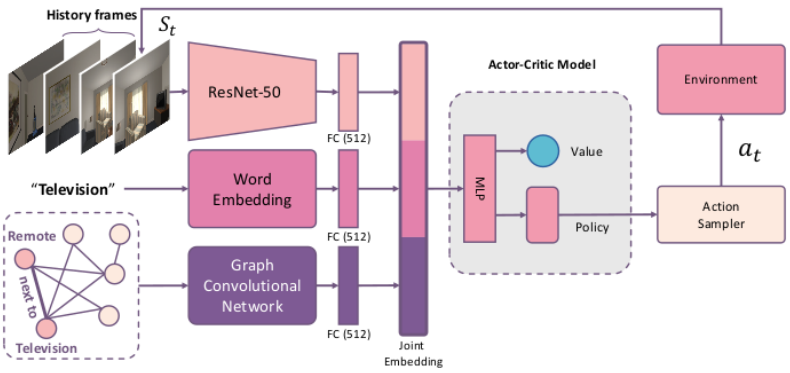


Figura 2.3: Ejemplo de sistema basado en objetivos visuales, texto e información semántica (imagen de [16]).

paro por episodio.

La principal desventaja es que este trabajo no reporta el equipo de cómputo utilizado, pero al observar el diagrama de la arquitectura se concluye que es un sistema que requiere mucho espacio de memoria ya que requiere de tres etapas de extracción de características, además de ejecutar instancias del simulador en paralelo. Otra desventaja es que la información semántica de las escenas se obtiene al preprocesar la información en el simulador fuera de línea para generar los grafos.

En 2017 Mirowski y Pascanu [21] proponen utilizar navegación basada en un objetivo usando aprendizaje reforzado, utilizando un simulador de ambientes tipo laberinto, en el cual existen algunos obstáculos dinámicos. Algunas escenas se muestran en la figura 2.4.

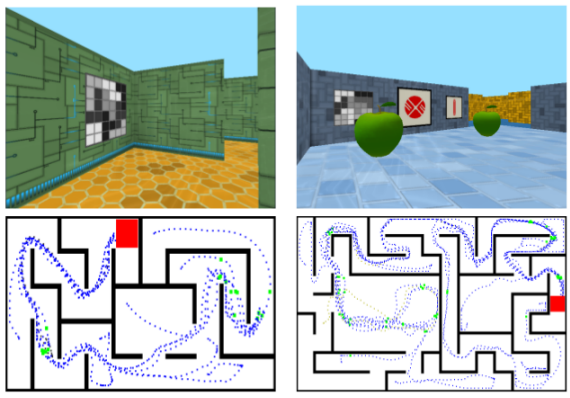


Figura 2.4: Escenas capturadas en el simulador tipo laberinto. El recuadro rojo indica el objetivo en el ambiente. Los puntos verdes indican un error en la estimación del lugar ya visitado. Los puntos azules indican la posición del agente (imagen tomada de [21]).

El objetivo de este trabajo es probar distintas tareas auxiliares para obtener información del ambiente, e integrarlas al aprendizaje en un agente que tiene por objetivo navegar a un lugar determinado. Estas tareas auxiliares son detectar que ya se ha visitado un lugar, agregar un módulo de predicción de la profundidad del ambiente, estimar la velocidad relativa. Todo esto con el fin de dar más información al sistema de aprendizaje sobre las consecuencias de ejecutar cierto tipo de acciones de movimiento.

Se utilizó un mecanismo de inferencia denominado *A3C* (del inglés *Asynchronous Advantage Actor-Critic*) el cual entrena un agente utilizando muchos hilos de ejecución para recolectar observaciones simultáneamente. Dicho método se basa en optimizar la probabilidad de una secuencia completa de transiciones entre estados.

Las arquitecturas propuestas se muestran en la figura 2.5. A continuación se listan las variaciones que describen los autores:

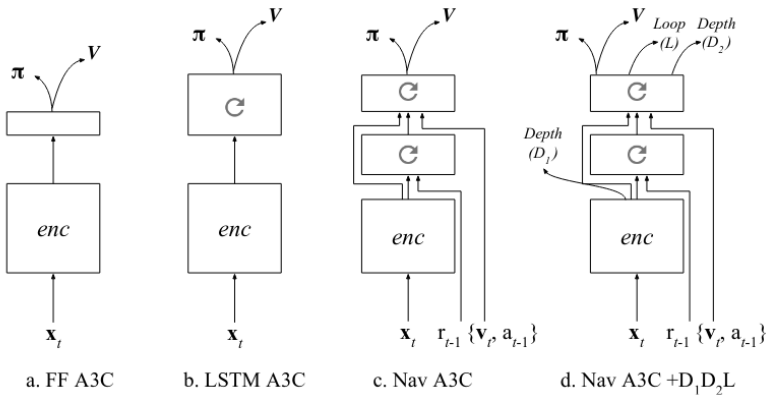


Figura 2.5: Ejemplo de arquitecturas usadas en [21]. La palabra *enc* se usa como abreviación en inglés de *encoder*. El símbolo de flecha con forma circular indica un bloque recurrente (imagen tomada de [21]).

1. FF-A3C como referencia (figura 2.5, el inciso a significa una capa convolucional para extraer características (*enc* como abreviación del inglés *encoder*) y un perceptrón multicapa para el cálculo de una función V y π. Para el caso de V se estima el valor de recompensa en promedio observando un numero finito de imágenes, y para π la probabilidad de ejecutar una acción a, dado un estado s.

2. LSTM+A3C, que es el mecanismo del número anterior, pero incluye un método de recurrencia denominado *unidades LSTM* (siglas del inglés *Long Short-Term Memory*) [22] (véase la figura 2.6) con el objetivo de incorporar dependencia temporal en el agente (figura 2.5, inciso b).

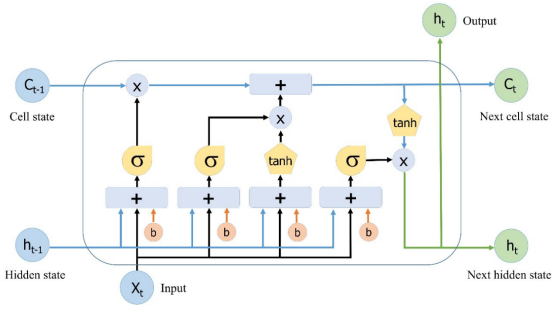


Figura 2.6: Celda LSTM (siglas del inglés *Long Short-Term Memory*). Es un tipo de celda que incorpora dependencia a corto y largo plazo mediante el uso de funciones no lineales como compuertas (imagen de tomada de [23]).

3. Nav A3C, que es una variación con celdas LSTM consecutivas, pero también incorpora la recompensa acumulada anterior r_{t-1}, la acción anterior ejecutada a_{t-1} y la velocidad relativa actual v_t del agente (figura 2.5, inciso c).

4. Nav A3C+$D_1 D_2 L$ en una extensión del inciso anterior pero D_1 se refiere a la predicción de profundidad a partir de la red convolucional. Para D_2 se hace lo mismo pero a partir de las salidas de las celdas LSTM. L es la estimación de estar en un lugar ya visitado usando la posición actual (figura 2.5, inciso d).

Otro trabajo relevante es el del año 2018 por Zhou [24]. En este se plantea entrenar dos sistemas de navegación, un planeador local y uno global, ambos usando aprendizaje reforzado.

El planeador global es construido usando la arquitectura de la figura 2.7, y su mecanismo de entrenamiento es A3C. Se usa una representación del destino con un vector de 1×4 dimensiones, con '0' y en sus entradas excepto en la que representa la clase de habitación a navegar, la cual tiene el valor '1'. Como observaciones del ambiente se usan imágenes de color con dimensión de 60×60 píxeles. Las imágenes se alimentan a una red convolucional nombrada *VGG-16* [25]. La captura de cada paso en el ambiente como la imagen del destino son procesadas con la red convolucional. Sus características se concatenan junto con un vector que representa la clase destino. Esto forma un vector de estado que se alimenta a un sistema de aprendizaje por refuerzo. Las salidas son las acciones a ejecutar predefinidas como: *adelante*, *izquierda*, y *derecha*.

El planeador global es entrenado bajo supervisión, mediante entrenamiento de secuencias de imágenes etiquetadas por un operador en ambientes estáticos (alrededor de 18 mil imágenes formando distintas trayectorias, dirigidas hacia cuatro habitaciones y

usando un mapa de celdas de ocupación). Cabe notar que los autores no anotan la rotación ni el avance de manera exacta, solo de manera semántica.

Para el caso del navegador local (ver figura 2.8), se entrena en un simulador nombrado Gazebo usando como entradas 36 lecturas de un sensor láser modelado. Dentro del simulador se modelan obstáculos dinámicos y se usa un método de aprendizaje por refuerzo denominado *aprendizaje Q*. Se entrena la ejecución de acciones como son: avanzar 0.03 m; girar izquierda o derecha en 60°; girar izquierda o derecha en 30°. El robot obtiene una recompensa de $r = 0.1$ en cualquier ejecución sin colisión, si colisiona se le da una recompensa negativa igual a $r = -20$, tanto para el planeador local como el global.

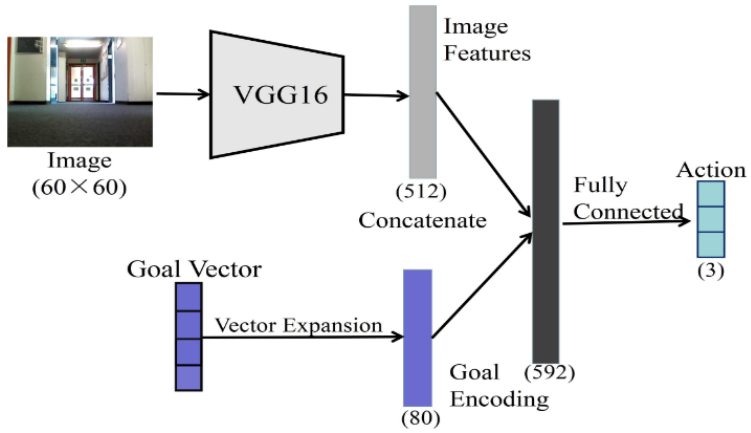


Figura 2.7: Navegador global (imagen tomada de [24]).

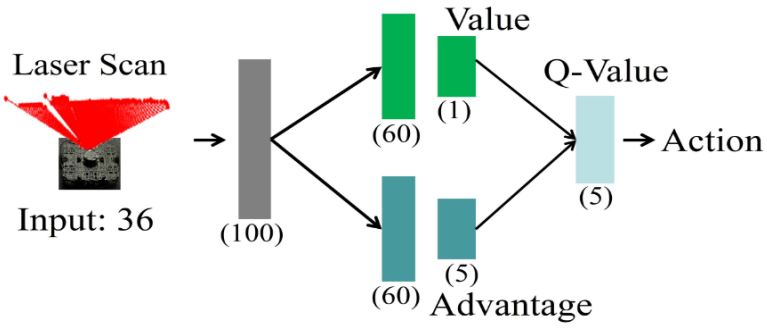


Figura 2.8: Navegador local (imagen tomada de [24]).

Al final, se combina el planeador global y el planeador local para la ejecución de la navegación en paralelo. Si los sensores láser detectan un obstáculo a cierta distancia se intercambia el planeador global por el local. Este enfoque resulta novedoso ya que es

posible entrenar un sistema de navegación de alto nivel y combinarlo con uno más especializado con control de bajo nivel. La principal desventaja es que los autores entrenan a su sistema de navegación global de manera completamente supervisada, limitando la cantidad de experiencia que puede obtener el robot, además el método A3C no incorpora ningún mecanismo de interpolación entre estados, ya que no fue diseñado con el enfoque de hacer a lo que se denomina *aprendizaje reforzado inverso*. A3C fue creado para desacoplar el entrenamiento entre varias instancias simuladas.

En la siguiente sección se comparan las características de los sistemas explicados anteriormente, con la finalidad de exponer la ventaja del sistema planteado en el capítulo 1 como propuesta de esta investigación.

2.2. Discusión

Anteriormente se describió como es posible usar localización y mapeo simultáneo (LMS) para estimar la posición de una cámara digital. También se describió como sistemas basados en aprendizaje por refuerzo (AR) pueden usar aprendizaje profundo para obtener un sistema completo de navegación.

En el caso de querer utilizar sistemas localización y mapeo simultáneo como solución, es necesario el incoporar estos sistemas como parte de sistemas más grandes en el control de un agente robótico. Un ejemplo de estos sistemas se muestra en [26] denomidado *VIRBOT*, el cual es una propuesta de arquitectura de los sistemas para controlar un agente robótico de servicio. En particular, la figura 2.9 muestra en la sección coloreada con color azul y color verde la conexión entre módulos, los de planeación y la navegación y los de control motriz respectivamente. Dicha conexión implica gran complejidad y una alta dependencia entre sistemas. Este problema de complejidad puede abordarse integrando sistemas de *entrada-salida* como es una red neuronal profunda que ejecuta tareas de detección o una red neuronal profunda que ejecute una política de ejecución de acciones directamente de sus entradas, como pueden ser imágenes del ambiente en el que se encuentra el agente. Este mecanismo no reemplaza sistemas tradicionales como planeación en celdas o en una representación de grafos, sin embargo se puede aportar un mecanismo de robustez e incluso ejecutarse de manera descentralizada mediante acceso a GPUs remotos o GPUs de propósito específico abordo del robot. Actualmente los sistemas basados en navegación utilizan sistemas LMS visuales revisados en [27].

Para aportar nuevos métodos de control de la ejecución de acciones en agentes robóticos, se han desarrollado avances en la optimización, eficiencia de datos y distintos

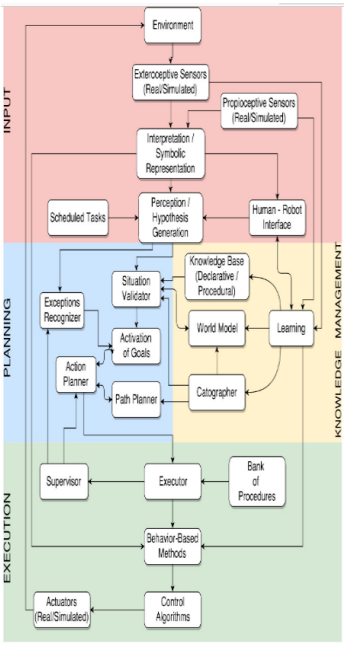


Figura 2.9: Arquitectura VIRBOT. Arquitectura propuesta en [26] para operar un robot de servicio. (imagen tomada de [26]).

tipos de fusión de información para utilizar aprendizaje reforzado como mecanismo para encontrar una secuencia de acciones para realizar la navegación. Algunos trabajos de investigación utilizan simuladores como sistemas de pruebas tanto para la evaluación de nuevos métodos de aprendizaje reforzado o nuevas arquitecturas, como para preentrenamiento para la posterior transferencia a robots reales. Muestra de estos dos enfoques son los trabajos de Mirowski [21] en el caso de evaluación de arquitecturas, como el trabajo de Zhou [24] para el caso de un robot real, con entrenamiento semi supervisado y por refuerzo. El trabajo de Wang y otros en [16] está más enfocado en la integración de información semántica y fotorrealista para futuros trabajos en la transferencia de modelos a agentes reales.

Es un hecho que la investigación en sistemas de visualización y el uso de nuevos procesadores de gráficos, que incluso recientemente ya incorporan procesadores de trazado de rayos, ha podido dar vida a entornos de desarrollo de videojuegos como son Unity. Este entorno de desarrollo incopora física avanzada y gráficos realistas usando trazado de rayos en hardware, para la creación de efectos de iluminación más reales. Por esto, en la actualidad ya es posible integrar estas herramientas para buscar nuevas soluciones al problema de navegación sin necesidad de tener un robot real para pruebas. Este es el

enfoque de este trabajo de investigación doctoral.

El sistema propuesto se muestra brevemente en la figura 2.10, pero se desarrolla a detalle en el capítulo 7. La propuesta está basada en el uso de imágenes fotorrealistas obtenidas dentro de un simulador para establecer el estado actual de un robot y su destino. Posteriormente se integra una memoria visual del ambiente en forma de grafo, como se hace en sistemas de navegación tradicionales pero sin el requerimiento de ser una representación densa y geométrica, que requiere mecanismos de óptimizacion complejos de resolver. Esta memoria solo captura la apariencia visual del entorno en un número reducido de tomas y se mostrará más adelante que mejora la obtención de una política de navegación más eficiente.

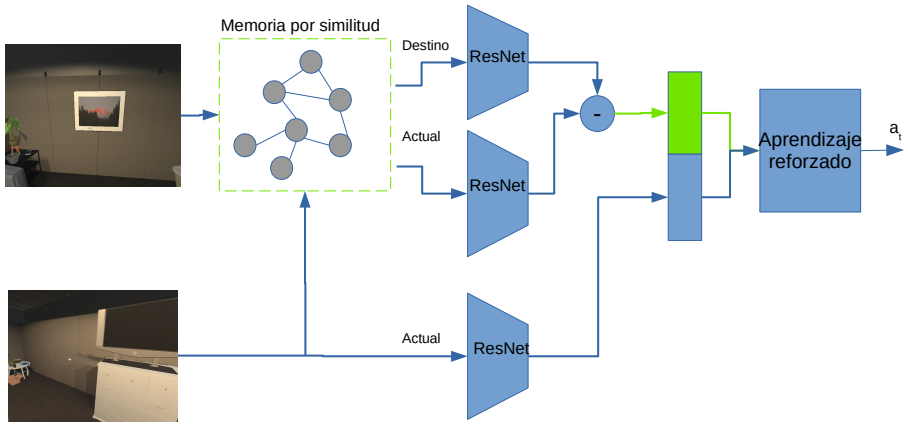


Figura 2.10: Diagrama de bloques del sistema propuesto.

Finalmente podemos resumir las ventajas de la propuesta, en los siguientes puntos:

- Sistema entrada-salida basado en aprendizaje reforzado.
  En comparación con sistemas localización y mapeo simultáneo (LMS) o rastreo y mapeo en paralelo (RMP) se sustituye la deteccion de puntos y sus rastreo, y todos los subsistemas por técnicas de aprendizaje profundo en un solo sistema.

- Sistema basado en la exploración del ambiente y condicionado visualmente.
  En sustitución de un mapa, se utiliza una red profunda como mecanismo de retención de experiencia y a su vez esta, puede seleccionar un conjunto de acciones de navegación. Esto evita la construcción de subsistemas de planeación que dependen de módulos LMS o RMP.

- El mecanismo de entrenamiento permite la fusión de fuentes de información. Permitiendo la incorporación, actualización o la expansión de su funcionalidad.

En el caso de sistemas LMS o RMP debe replantearse el como agregar información semántica o de otros tipos de sensores.

En el capítulo 3 se presenta la teoría para poder aplicar aprendizaje por refuerzo para resolver la tarea de navegación, utilizando solo imágenes.

Capítulo 3

Procesos de Decisión de Markov

En este capítulo se describe la teoría relacionada con el *aprendizaje por refuerzo* (AR) y como se aplica a sistemas utilizando un enfoque de aprendizaje por *ensayo y error*. Si se tiene un sistema robótico que interactúa con un ambiente real o simulado es posible inferir una secuencia de acciones con mayor utilidad en un periodo de tiempo, esto permite establecer relaciones entre una señal de recompensa acumulada, obtenida de un conjunto de acciones y estados observados en el ambiente.

La sección 3.1 y la 3.2 establece los elementos teóricos de un *proceso de decisión de Markov* (PDM), la sección 3.4 describe la herramienta matemática para resolver este problema mediante interacciones secuenciales con el ambiente.

3.1. Procesos de Markov

Un *proceso de Markov* (PM) es un proceso estocástico en el cual una secuencia aleatoria de estados $\{s_{t=1}, s_{t=2}, ..., s_{t=T}\}$ cumple con la propiedad Markoviana. Esto es que, dado el par $\{S, \mathbf{P}\}$ donde S es un conjunto finito de T estados. Y $\mathbf{P}$ una matriz con la probabilidad de transición entre pares $(s_{i=t}, s_{j=t+1})$ se define la probabilidad $P_{ss'}$ entre estados como:

$$P_{ss'} = p(s_{t+1} = s' \mid s_t = s) \tag{3.1}$$

Este proceso estocástico es conocido como una *cadena de Markov de primer orden*. El orden indica la dependencia temporal de los estados anteriores. En este caso, solo el estado anterior.

Esta propiedad se muestra en la figura 3.1.

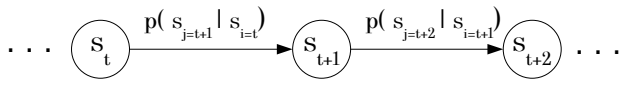


Figura 3.1: Esquema de la propiedad de Markov. Las flechas representan la probabilidad de observar el estado s_j, dado que se observa el estado anterior s_i.

Si se desea modelar la ejecución de acciones sobre estados observados secuencialmente en el tiempo, y bajo la restricción de un criterio de selección, debemos extender la definición del PM agregando un conjunto finito para la acciones posibles. Además, se requiere una función de evaluación sobre las acciones ejecutadas.

En la siguiente sección se describe dicha esta extensión matemática de los procesos de Markov.

3.2. Procesos de decisión de Markov

Un *proceso de decisión de Markov* (PDM) se define como la 5-tupla $\{S, A, \mathbf{P}, R, \gamma\}$. En donde S y A son conjuntos finitos. Un estado $s_t \in S$ es un estado en un ambiente que se puede medir, y $a_t \in A$ es una acción posible. La matriz $\mathbf{P}$ es una matriz con probabilidades de transición entre estados. El valor $r_{t+1} \in R$ es un valor de recompensa estimado por alguna función al ejecutar una acción a_t. Cabe notar que s_t y a_t pueden ser discretos o continuos. Finalmente, el parámetro $\gamma \in [0, 1)$ es un factor de penalización temporal para garantizar la convergencia cuando $t- > \infty$.

La dinámica en un PDM se define por la probabilidad de transitar a un estado siguiente, dada la configuración de estado anterior y la acción ejecutada. Esto se define como:

$$P^a_{ss'r} = p(s_{t+1} = s', r_{t+1} = r \mid s_t = s, a_t = a) \tag{3.2}$$

La distribución de probabilidad para cada estado $s, s' \in S$ y cada acción posible $a \in A$ cumple que:

$$\sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a) = 1 \tag{3.3}$$

La ecuación 3.3 nos muestra que si conocemos la probabilidad para todos los valores de s' y r, dados s y a, entonces podemos plantear cualquier relación de transición en el ambiente. Por ejemplo, podemos definir la probabilidad de transición entre estados solo

en función de la acción ejecutada. Esto es:

$$P^a_{ss'} = p(s_{t+1} = s' \mid s_t = s, a_t = a) = \sum_{r \in R} p(s', r \mid s, a) \tag{3.4}$$

También podemos definir el valor esperado de la recompensa para cada par estado-acción como una función $r(s, a)$, esto es:

$$r(s, a) = \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a] = \sum_{r \in R} \sum_{s' \in S} p(s', r \mid s, a) \tag{3.5}$$

La figura 3.2 muestra la dependencia de probabilidades para transitar a estados siguientes s_{t+1}, dada una acción a_t. En particular, se muestra expresada la probabilidad $p(a_t \mid s_t)$ como una función $\pi_\theta(a_t \mid s_t)$ que es una parametrización sobre los valores θ. En la siguiente sección se explicará como solucionar este problema sin conocer directamente las probabilidades de transición.

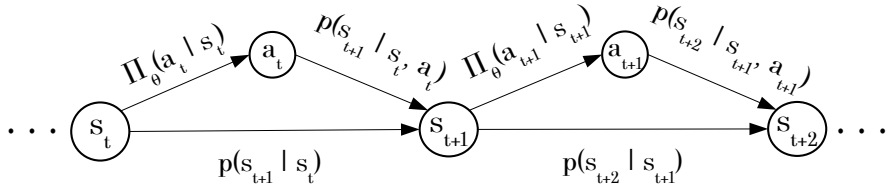


Figura 3.2: Modelo gráfico para un PDM. La función $\pi_\theta(a_t \mid s_t)$ es una parametrización común.

Un PDM modela el problema de la interacción con un medio ambiente, en el cual es posible estimar el cambio entre estados conforme el tiempo avanza.

Se define a un *agente* como un proceso en el cual se asocia una representación de estados con un valor de salida mediante una acción sobre el entorno, el valor de salida permite seleccionar una acción que se desea ejecutar al siguiente periodo de tiempo. El *ambiente* es cualquier sistema con el que se interactúa y se toma una medición [13]. Esta relación entre un agente y un ambiente se muestra en la figura 3.3 .

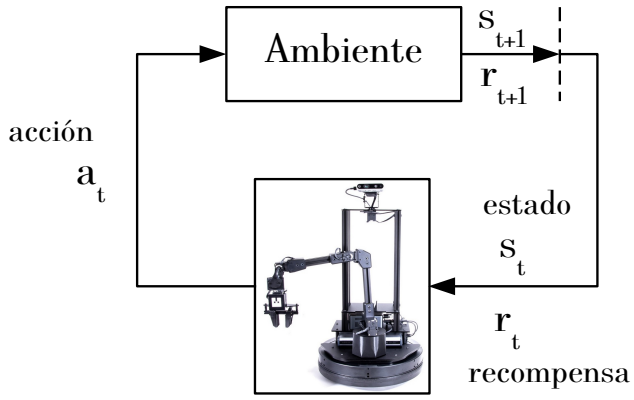


Figura 3.3: Interacción agente-ambiente en un PDM. El agente es representado por el robot LocoBot [28].

El agente y el ambiente interactúan secuencialmente, el agente selecciona una acción y la ejecuta, modificando el estado actual. Después, el ambiente es observado y se obtiene un valor de recompensa que el agente trata de maximizar sobre un periodo de tiempo.

Para el caso de este trabajo, el tiempo es una secuencia discreta de pasos, $t = 0, 1, 2, ...$, en los cuales el agente recibe una representación s_t y ejecuta una acción a_t, posteriormente recibe una recompensa r_{t+1}. Esto lleva al agente a una nueva configuración representada por s_{t+1}.

El PDM y el agente dan lugar a una secuencia o trayectoria de la forma:

$$\{s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, ...\} \tag{3.6}$$

Si expresamos las recompensas recibidas por el agente después de un periodo de tiempo como $r_{t+1}, r_{t+2}, r_{t+3}, ...$, entonces es factible maximizar esta secuencia de valores mediante una función. La forma más simple de definir una función es usando la suma de las recompensas por cada acción ejecutada, en un horizonte temporal T. Esto es:

$$g_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{3.7}$$

El valor g_t se conoce como *recompensa acumulada*. La valor de la variable T indica que se ha terminado la ejecución alcanzando un *estado terminal* s_T. El número T es una variable aleatoria ya que no se sabe cuando el agente alcanzará un objetivo o terminará la ejecución en estados no deseados (colisión o una acción no útil).

En ciertos casos, no es posible determinar un estado terminal por lo que se puede generalizar a g_t haciendo $T = \infty$ (usando una notación matemática no estricta), y agregando

una constante $\gamma < 1$ para establecer un valor de recompensa acumulado finito. Esto es:

$$g_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ... = \sum_{k=0}^{T=\infty} \gamma^k r_{t+k+1} \tag{3.8}$$

Podemos notar de la ecuación 3.8 lo siguiente:

$$g_t = r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + ...) \tag{3.9}$$

Por lo que al reescribir dicha ecuación como una función recurrente, toma la forma:

$$g_t = r_{t+1} + \gamma g_{t+1} \tag{3.10}$$

Y en términos de las recompensas r_t, se tiene la forma:

$$g_t = r_{t+1} + \sum_{k=1}^{T=\infty} \gamma^k r_{t+k+1} \tag{3.11}$$

La recompensa acumulada ahora se nombra como *recompensa acumulada con descuento*, y toma un valor mínimo si $\gamma = 0$ y se selecciona el menor valor $r_{t+1} \in R$ en cada instante; es máxima si $0 < \gamma < 1$ y se selecciona el mayor valor $r_{t+1} \in R$ en cada tiempo. Esto acota los valores posibles de recompensa aún en episodios sin fin. Esto es:

$$2 \text{ mín}\{R; \gamma = 0\} \leq g_t \leq \frac{1}{1-\gamma} \text{máx}\{R; 0 < \gamma < 1\} \tag{3.12}$$

En la práctica, se restringe el aprendizaje a episodios finitos debido a límites de almacenamiento y cómputo, por lo que la ecuación 3.8 se reescribe para un tiempo finito T como:

$$g_t = r_{t+1} + \sum_{k=1}^{T} \gamma^k r_{t+k+1} \tag{3.13}$$

3.3. Solución a un proceso de decisión de Markov.

Para resolver el problema de seleccionar la mejor acción a ejecutar sobre un proceso de decisión de Markov (PDM), debemos encontrar una función que mapea el estado actual sobre cada posible acción. Esta función de mapeo se conoce como *política de acción* o por brevedad solo *política*. Se denota como una función $\pi(\ )$.

Si $\pi(s) = a$ genera un único valor para cada s, entonces se dice que π es una *política determinística*. Si π es una función de probabilidad de la forma $\pi(a \mid s)$ entonces es una *política estocástica*, la cual se supone una distribución estacionaria. Esto es $a \sim \pi(a_t \mid s_t), \forall t > 0$.

Es común definir funciones auxiliares para estimar cantidades útiles sobre el PDM. Se define una *función valor-estado* $v_\pi(\ )$ como el valor esperado de recompensa dado un estado s y una política π. Esto se expresa como:

$$v_\pi(s) = \mathbb{E}_\pi [g_t \mid s_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid s_t = s \right], \forall s \in S \tag{3.14}$$

Similarmente, se puede definir una *función valor-acción* $q_\pi(\ )$ que regrese el valor esperado de recompensa al ejecutar una acción a, con una política de acción π. Es decir:

$$q_\pi(s, a) = \mathbb{E}_\pi [g_t \mid s_t = s, a_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right] \tag{3.15}$$

En la siguiente sección se explicará como aplicar las ecuaciones 3.14 y 3.15 al problema de aprendizaje reforzado usando métodos de aproximación iterativos.

3.4. Aprendizaje Reforzado

El *aprendizaje reforzado* (AR) se refiere a un tipo de algoritmo de aprendizaje de máquina en el que un agente recibe una recompensa acumulada, con un retardo temporal respecto del momento en que se ejecutó una acción o comando. Esto es debido a que se requiere de una secuencia de ejecuciones sobre un ambiente y después deben ser evaluadas usando una política de acción. La política puede ser ajustada en cada instante o después de un periodo de tiempo.

También, se requiere de un medio ambiente con el que se pueda interactuar, y que sea posible observar cambios en él a través de mediciones (medir transiciones de estados). Este método se enfoca en un desempeño general sobre un objetivo, balanceando acciones de exploración y su incorporación a una modelo de conocimiento.

Para resolver el problema de AR, dado un PDM, se establece el conjunto $\{S, A, \mathbf{P}_\pi.R_\pi, \gamma\}$. En donde ahora $\mathbf{P}$ y R tienen dependencia del criterio de acción en cada estado π.

Ya que existen muchas maneras de seleccionar acciones, dada una configuración actual del ambiente s, debemos establecer una *política óptima* denotada como π^*. La política óptima debe maximizar el valor de recompensa $v_\pi(s)$ sobre cualquier acción ejecutada, y esto se logra seleccionando las transiciones de estados que regresan el mayor valor

$r_{t+1} \in R_\pi$. Esto a su vez implica maximizar $q_\pi(s,a)$, por lo que a π se le nombra como una *política voraz*. Con este fin, definimos las funciones auxiliares de valor-estado $v_\pi(s)$ y valor-acción $q_\pi(s,a)$ dada una política π, como óptimas si regresan el mayor valor de recompensa acumulada.

Se define a *una función óptima valor-estado* $v^*(s)$ como:

$$v^*_\pi(s) = Max_\pi [v_\pi(s)] \tag{3.16}$$

Similarmente, se define una *función óptima valor-acción* $q^*(s,a)$ como:

$$q^*_\pi(s,a) = Max_\pi [q_\pi(s,a)] \tag{3.17}$$

Supongamos que tenemos un par ordenado de políticas π y π' y se sabe que π obtiene mejores valores r_{t+1} que π', entonces:

$$\pi \geq \pi' \; si \; v_\pi(s) \geq v'_\pi(s), \forall s \tag{3.18}$$

Entonces existe una política óptima π^* si:

$$\pi^* \geq \pi, \forall \pi \tag{3.19}$$

Lo que lleva a la conclusión de que π^* alcanza valores-estado óptimos $v^*_\pi(s)$ en donde:

$$v^*_\pi(s) \geq v_\pi(s), \; \forall \pi \tag{3.20}$$

Esto se extiende a valores-acción óptimos q^*_π con:

$$q^*_\pi(s,a) \geq q_\pi(s,a), \; \forall \pi \tag{3.21}$$

En el caso de requerir una *política óptima* estocástica $\pi^*(a \mid s)$, se puede obtener mediante la ecuación siguiente:

$$\pi^*(a \mid s) = \begin{cases} 1, \; a = Arg\,\text{máx}\,[q^*(s,a)] \\ 0, \; en\ otro\ caso \end{cases} \tag{3.22}$$

Si se conoce $q^*_\pi(s,a)$ en cada tiempo, entonces se conoce $v^*_\pi(s)$, lo que determina la política óptima. Entonces el PDM esta resuelto con π^* entrenada, y se puede decidir que acción ejecutar en cualquier momento.

3.4.1. Métodos de solución para aprendizaje reforzado.

A continuación se enlistan los cuatro enfoques más comunes para resolver un PDM usando AR:

- **Soluciones basadas en muestreo.**
  Dado un conjunto de entrenamiento $\mathbf{D} = \{(s, a, r, s')_t\}_{t=1}^{N}$ recolectado del ambiente con N interacciones, se puede aplicar lo siguiente:

  - AR basado en modelo.
    - Se establece y refina un modelo para $P(s' \mid s, a)$ y $r(s, a)$ mediante $v(s)$. Al final se obtiene π^* restringida a la ecuación 3.3.
  - AR sin modelo.
    - Se estima una función $v_\pi(s)$ por aprendizaje, codificando una política parametrizada que cumple $\pi_\theta \approx \pi^*$.
    - Se optimiza directamente una política parametrizada π_θ iterando sobre $\mathbf{D}$.

- **Soluciones basadas en demostraciones.**
  Dado un conjunto de entrenamiento $\mathbf{D} = \{(s_{0:T}, a_{0:T})_d\}_{d=1}^{n}$ con n ejemplos:

  - Aprendizaje por imitación.
    - Se aprende $\pi(s)$ directamente.
  - AR inverso.
    - Se aprende una función $r(s, a)$ *latente* (es decir, supervisada) y al estimar $v_\pi(s)$ se obtiene $\pi(s)$.

En este trabajo se aborda la aplicación de *aprendizaje reforzado sin modelo*. En la siguiente sección se explicarán los detalles de este método de solución, aplicado a la navegación visual de un agente.

3.4.2. Un enfoque sin modelo. Método de diferencias temporales.

Existen distintos métodos de solución para obtener las transiciones en un PDM usando AR. Un resumen para todas las variaciones de solución se discute en [13], usando modelos

para $P^a_{ss'}$, y métodos iterativos sin modelo. En esta sección solo se expone la solución sin modelo, por el método de diferencias temporales.

Primeramente, redefinamos la función valor-estado $v_\pi(s)$ (ecuación 3.14) sujeta a una política π de manera recurrente. Esto es:

$$v_\pi(s) = \mathbb{E}_\pi \left[r_{t+1} + \gamma v_\pi(s_{t+1} = s') \mid s_t = s \right] \tag{3.23}$$

Igualmente para la función de valor-acción $q_\pi(s,a)$ (ecuación 3.15), se tiene:

$$q_\pi(s,a) = \mathbb{E}_\pi \left[r_{t+1} + \gamma q_\pi(s_{t+1} = s', a_{t+1} = a') \mid s_t = s, a_t = a \right] \tag{3.24}$$

El método establece aproximar π^* iterativamente observando secuencias $\{s,a,r,s',a'\}$. Además, se puede considerar una política estocástica $\pi_\theta(s)$ con parámetros θ, que aproxime la distribución de $P^a_{ss'}$. Para esto, se propone que la función de valor-estado sea $v_\pi(s) \approx \hat{v}_\pi(s)$, en la cual se actualiza $\hat{v}_\pi(s)$ en función de valores temporales anteriores y de una actualización progresiva, usando una constante de aprendizaje $\alpha \in (0,1)$ [1] [13].

La aproximación se define como:

$$\hat{v}_{\pi,i+1}(s) \leftarrow (1-\alpha)\hat{v}_{\pi,i}(s) + \alpha \left[r + \gamma \hat{v}_{\pi,i}(s') \right] \tag{3.25}$$

La ecuación 3.25 se conoce como el método *TD* (del inglés *Temporal Difference*). Si tomamos la acción óptima en un conjunto de transiciones, dicha ecuación se transforma como:

$$\hat{v}_{\pi,i+1}(s) \leftarrow (1-\alpha)\hat{v}_{\pi,i}(s) + \alpha \left[r + \gamma \max_{s' \in \mathbf{S}} \{\hat{v}_{\pi,i}(s')\} \right] \tag{3.26}$$

Para la función valor-acción, se aplica el mismo razonamiento, lo que da lugar a la recursión de $q_\pi(s,a) \approx \hat{q}_{\pi,i}(s,a)$, esto se expresa de la forma:

$$\hat{q}_{\pi,i+1}(s,a) \leftarrow (1-\alpha)\hat{q}_{\pi,i}(s,a) + \alpha \left[r + \gamma \hat{q}_{\pi,i}(s',a') \right] \tag{3.27}$$

La ecuación 3.27 se conoce como el método *SARSA*, debido a la secuencia $\{s,a,r,s',a'\}$ que debe ser almacenada para evaluar $\hat{q}_\pi$. También se le denomina como un método de *política en linea* (una acepción del inglés *on-policy*) ya que se usa la misma política durante la actualización [2].

Si se aplica el criterio de maximizar la recompensa en la ecuación anterior, y se agrega

[1] $\hat{v}_\pi \to v_*$, *si* $\{s,a,r,s',a'\}_{t=1}^n$ *y* $n \to \infty$.

[2] $\hat{q}_{\pi,i}(s,a)$ y $\hat{q}_{\pi,i}(s',a')$ son la misma función con distintos valores de entrada en la iteración i.

una política estocástica π_θ, se obtiene:

$$\hat{q}^*_{\pi,i+1}(s,a) \leftarrow (1-\alpha)\, \hat{q}^*_{\pi,i}(s,a) + \alpha \left[r + \gamma \max_{a' \in \mathbf{A}} \left\{ \hat{q}^*_{\pi,i}(s', \pi_\theta(s')) \right\} \right] \tag{3.28}$$

La ecuación 3.28 se conoce como el método *aprendizaje Q* [12]. Dado que la política π_θ puede ser cualquier tipo de selección, y el criterio de máxima recompensa elige la mejor política para actualizar $\hat{q}^*_\pi$, a este método se le denomina como una *política fuera de linea*.

Es una práctica común usar una política estocástica definida como:

$$\pi_\theta(a \mid s) = \frac{e^{f_\theta(s,a)}}{\sum\limits_{a' \in \mathbf{A}} e^{f_\theta(s,a')}} \tag{3.29}$$

Donde f_θ es una función no lineal con parámetros $\theta_i \in \theta$.

3.4.3. Aprendizaje reforzado usando aprendizaje profundo.

En la actualidad es una práctica común parametrizar a la función valor-estado $\hat{v}_\pi(s)$ como una función $f_{\theta_v}(s)$ para facilitar su optimización. También es común usar técnicas de gradiente descendente. La regla de actualización para $f_{\theta_v}(s)$, para cada parámetro θ_i usando gradiente descendente se define como:

$$\theta_{v,i+1} \leftarrow \theta_{v,i} + \alpha \Big[r + \gamma f_{\theta_v}(s') - f_{\theta_v}(s)) \Big] \nabla_{v,\theta_i} f_{\theta_v}(s) \tag{3.30}$$

Para encontrar $\hat{q}_\pi(s,a)$, se sigue el mismo procedimiento que con $\hat{v}_\pi(s)$ pero sobre una función $f_{\theta_q}(s,a)$. Esto se expresa como:

$$\theta_{q,i+1} \leftarrow \theta_{q,i} + \alpha \left[r + \gamma \max_{a' \in \mathbf{A}} \Big[f_{\theta_q}(s', a') \Big] - f_{\theta_q}(s,a)) \right] \nabla_{q,\theta_i} f_{\theta_q}(s,a) \tag{3.31}$$

En donde $\alpha, \gamma \in (0,1)$ para la ecuaciones 3.30 y 3.31.

Si f_{θ_v} y f_{θ_q} son funciones no lineales obtenidas al usar un perceptrón multicapa, entonces este método es una mezcla de aprendizaje profundo y aprendizaje reforzado. Esta mezcla es una práctica común para encontrar políticas de acción en problemas de navegación autónoma.

3.4.4. Políticas basadas en gradientes.

Es posible optimizar el valor esperado de recompensa, dada una secuencia temporal de transiciones entre estados, con su respectiva recompensa r_t.

Sea $\tau = \{s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1}, s_t, a_t\}$, entonces la probabilidad $p(s_t, a_t)$, dada $\pi_\theta(a_t \mid s_t)$ y $p(s_t \mid s_{t-1}, a_{t-1})$ es (véase la figura 3.2):

$$p(s_t, a_t \mid \tau) = p(s_0) \prod_{i=1}^{t-1} \pi_\theta(a_i \mid s_{i-1}) p(s_i \mid s_{i-1}, a_{i-1}) \tag{3.32}$$

Se aplica la función logaritmo natural a ambos lados de la ecuación anterior y se tiene:

$$\log[p(s_t, a_t \mid \tau)] = \log[p(s_0)] + \sum_{i=1}^{t-1} \log[\pi_\theta(a_i \mid s_{i-1})] + \sum_{i=1}^{t-1} \log[p(s_i \mid s_{i-1}, a_{i-1})] \tag{3.33}$$

Y el gradiente de $\log[p(s_t, a_t \mid \tau)]$ respecto del parámetro θ es:

$$\nabla_\theta \log[p(s_t, a_t \mid \tau)] = \sum_{i=1}^{t-1} \nabla_\theta \log[\pi_\theta(a_i \mid s_{i-1})] \tag{3.34}$$

Ahora, se pretende optimizar una red neuronal con parámetros θ que implementa la política π_θ y que optimice el valor acumulado de recompensa. Entonces su función de perdida queda definida como:

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1} \mid \pi_\theta] = \sum_{t=0}^{T-1} p(s_t, a_t \mid \tau) r_{t+1} \tag{3.35}$$

Por lo que el gradiente $\nabla_\theta J(\theta)$ es:

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} r_{t+1} \nabla_\theta p(s_t, a_t \mid \tau) = \sum_{t=0}^{T-1} r_{t+1} (\sum_{i=1}^{t-1} \nabla_\theta \log[\pi_\theta(a_i \mid s_{i-1})]) \tag{3.36}$$

$$\nabla_\theta J(\theta) = g_t \sum_{i=1}^{t-1} \nabla_\theta \log[\pi_\theta(a_i \mid s_{i-1})], \forall s \in \mathbf{S} \tag{3.37}$$

Con π_θ diferenciable.

Si no se conoce el número total de estados y acciones como es el caso en el que estos son de naturaleza continua, el gradiente anterior no se conoce exactamente, entonces $\nabla_\theta J$ es una muestra del gradiente real. Por lo que se comporta como un estimador insesgado de la ecuación 3.37. Es decir:

$$\nabla_\theta J(\theta) \approx g_t \sum_{i=1}^{t-1} \nabla_\theta \log[\pi_\theta(a_i \mid s_{i-1})] \tag{3.38}$$

Un método para mantener la convergencia del gradiente es usar un factor de descuento temporal sobre g_t mediante una constante $\gamma \in (0,1)$, y restarle alguna función $b(t)$. Esta función es un promedio móvil de recompensas, en algún intervalo con k muestras [29, 30]. Esto es:

$$\nabla_\theta J(\theta) \approx \left[\sum_{t=0}^{k-1} \gamma^i r_{t+i} - b(t)\right] \sum_{i=1}^{t-1} \nabla_\theta log\left[\pi_\theta(a_i \mid s_{i-1})\right] \tag{3.39}$$

$$b(t) = \frac{1}{k} \sum_{t=0}^{k-1} r_t \tag{3.40}$$

La política ahora considera todas las transiciones entre estados, por lo que al tener muchos episodios de entrenamiento el gradiente se acerca a una política óptima de acción.

3.5. Navegación como un proceso de Markov

En este capítulo se expusó la teoría relacionada para resolver un proceso de decisión de Markov (PDM) condicionado a una señal de recompensa. Esta teoría permite modelar el problema de navegar en un ambiente como el encontrar una función que estime la secuencia de estados, acciones y recompensas $\{s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, ...\}$ en cada instante que se ejecute dicha función. Se ha desmostrado que este problema puede resolverse utilizando redes neuronales artificiales como estimadores de la función que aproxima una política de acción. Esto se describió matemáticamente en la ecuación 3.30, la ecuación 3.31, o en las ecuaciones 3.39 y 3.40. Para obtener resultados útiles se requiere de muchos episodios, debido a limitantes actuales en los métodos de optimización sin un modelo previo del ambiente o de la dinámica en el mismo. Esto justamente es la mayor ventaja de la teoría descrita en las secciones de este capítulo, no se requiere de un modelo de transiciones entre estados observados, pero trae consigo dificultades técnicas en la recopilación de suficientes imágenes para obtener la política de ejecución de acciones.

En el siguiente capítulo se presenta la teoría matemática para representar imágenes con un modelo probabilístico llamado *modelo oculto de Markov*, este modelo permite comparar imágenes respecto de una representación visual del ambiente y esto permite asociar imágenes no observadas a una representación del ambiente contruida previamente. A esta representación se le denomina *memoria visual por similitud estadística* y se usará como un mecanismo auxiliar para entrenar un agente mediante aprendizaje reforzado en la tarea de navegación visual.

Capítulo 4

Modelos Ocultos de Markov

En este capítulo se describe el primero de dos métodos utilizados para construir una memoria visual topológica, utilizando un modelo estadístico de la apariencia en dos imágenes distintas. El segundo método se describirá en el capítulo 5 y se describirá en el capítulo 6 el como se construye un grafo como una base de datos de similitud entre imágenes. El objetivo de usar un método estadístico es saber cual es la imagen más semejante a una de consulta, para poder agregar aristas nuevas a un vértice. Para esto, se utiliza un modelo probabilístico denominado modelo oculto de Markov (MOM).

4.1. Definición de un modelo oculto de Markov

Un *modelo oculto de Markov* (MOM) es un modelo de un *proceso estocástico* que se plantea como un proceso estocástico doble, en el cual las realizaciones de un primer proceso llamado *proceso oculto*, dan origen a un segundo proceso llamado *proceso observado*, los dos procesos estocásticos se logran caracterizar usando sólo el proceso que se puede observar (medir). El principal atributo de los modelos ocultos de Markov es que son modelos que asignan probabilidades de aparición a secuencias de símbolos, se dice que son *procesos generativos* ya que las probabilidades de ocurrencia para cada símbolo, se definen por una serie de pasos que incrementalmente producen la secuencia observada. Este tipo de proceso estocástico se le describe mediante un conjunto de unidades llamadas *estados* que son el mecanismo de emisión de los símbolos. Los estados tienen probabilidades de transición asociadas a los cambios internos del sistema durante la generación de los símbolos.

4.2. Arquitectura de un modelo oculto de Markov

Un *modelo oculto de Markov* (MOM) es un proceso estocástico que consta de un proceso de Markov no observado $S = \{S_t\}$, y un proceso observado $O = \{O_t\}$, con estados dependientes de los estados no observados [31]. Un MOM, discreto, de primer orden se define como $\lambda = (V, S, A, B, \Pi)$ donde:

- $V = \{v_1, v_2, ..., v_M\}$ es un alfabeto finito con M símbolos.
- $S = \{S_1, S_2, ..., S_N\}$ es un conjunto finito de N estados, con el estado al tiempo t denotado como q_t.
- $A = [a_{ij}]_{N \times N}$ es una matriz de probabilidades de transición donde el elemento a_{ij} es la probabilidad de transición del estado i al estado j.

  $a_{ij} = P(q_{t+1} = S_j \mid q_t = S_i)$ para todo $1 \leq i, j \leq N$.
- $B = [b_j(k)]_{N \times M}$ es la matriz de probabilidad de emisiónes, esto es que $b_j(k) = P(V_k \; en \; t \mid q_t = S_i)$.
- $\Pi = [\pi_i]_{1 \times N}$ es la matriz de probabilidades iniciales donde $\pi_i = P(q_1 = S_i)$ con $1 \leq i \leq N$.

Además $O = \{O_1, O_2, ..., O_T\}$ es la única secuencia que es posible observar, tiene longitud T y cada elemento O_t es un símbolo de los posibles en el conjunto V. Por conveniencia, se usa la notación compacta $\lambda = (A, B, \Pi)$ para referirse a un modelo oculto de Markov.

Un modelo oculto de Markov se representa como un grafo dirigido de transiciones y emisiones. La arquitectura que permita modelar de la mejor forma posible las propiedades observadas, depende en gran medida de las características del problema. Las arquitecturas más usadas son:

- *Ergódicas* o completamente conectadas. En estas cada estado del modelo puede ser alcanzado desde cualquier otro estado en un número finito de pasos. Un ejemplo se muestra en la figura 4.1.
- *Izquierda-derecha*, hacia adelante o Bakis. Estas tienen la propiedad de que en la medida de que el tiempo crece, se avanza en la secuencia de observación asociada O. En reconocimiento de la voz estas arquitecturas modelan bien la relación temporal entre los sonidos. Un ejemplo se muestra en la figura 4.2

- *Izquierda-derecha paralelas*. Estas son dos arquitecturas izquierda-derecha conectadas entre sí (ver figura 4.3).

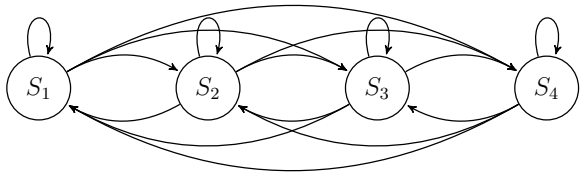


Figura 4.1: Ejemplo de modelo ergódico para cuatro estados.

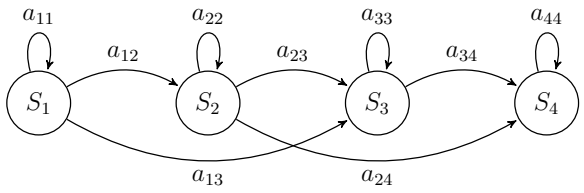


Figura 4.2: Ejemplo de modelo izquierda-derecha con cuatro estados.

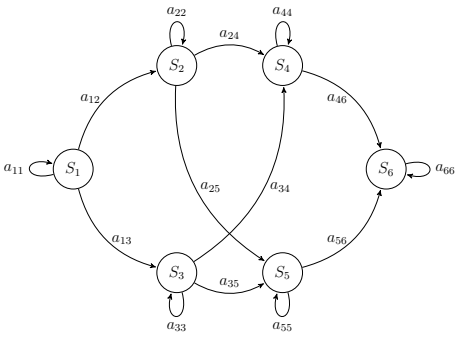


Figura 4.3: Ejemplo de modelo paralelo con seis estados.

4.3. Los tres problemas fundamentales

Existen tres problemas fundamentales que deben resolverse antes de poder aplicar los modelos ocultos de Markov en sistemas de reconocimiento de secuencias [31]. A continuación se enumeran:

1. Calcular la probabilidad de una secuencia observada O, se calcula $P(O \mid \lambda)$.
   Dada una secuencia de observaciones $O = \{O_1, O_2, ..., O_T\}$ y un modelo $\lambda = (A, B, \Pi)$.

2. Calcular la secuencia óptima de estados que genera las observaciones.
   Dada una secuencia de observaciones $O = \{O_1, O_2, ..., O_T\}$ y un modelo $\lambda = (A, B, \Pi)$, encontrar $Q = \{q_1, q_2, ..., q_T\}$ óptima para generar O.

3. El aprendizaje del modelo.
   Ajustar los parámetros A, B y Π para maximizar $P(O \mid \lambda_k)\ \forall \in \lambda_k$.

4.3.1. Solución al problema de la evaluación

Se supone que se tiene la secuencia fija de estados $Q = \{q_1, q_2, ..., q_T\}$ con q_1 el estado inicial. Entonces la probabilidad de que la secuencia O sea emitida por Q y un modelo oculto de Markov λ se expresa como

$$P(O \mid Q, \lambda) = \prod_{t=1}^{T} P(O_t \mid q_t, \lambda) \tag{4.1}$$

Asumiendo independencia entre las observaciones tenemos:

$$P(O \mid Q, \lambda) = \prod_{t=1}^{T} b_{q_t}(O_t) \tag{4.2}$$

Así, la secuencia de estados Q se expresa como:

$$P(Q \mid \lambda) = \pi_{q_1} \prod_{t=2}^{T} a_{q_{t-1} q_t} \tag{4.3}$$

Y la probabilidad conjunta de que O y Q ocurran es:

$$P(O, Q \mid \lambda) = P(Q \mid \lambda) P(O \mid Q, \lambda) = \pi_{q_1} \prod_{t=2}^{T} a_{q_{t-1} q_t} \prod_{t=1}^{T} b_{q_t}(O_t) \tag{4.4}$$

Si se consideran todas las secuencias de estados posibles se tiene que

$$P(O \mid \lambda) = \sum_{\forall q_i} \pi_{q_1} b_{q_1 O_1} \prod_{t=2}^{T} a_{q_{t-1} q_t} b_{q_t O_t} \tag{4.5}$$

La ecuación 4.5 se conoce como *evaluación por fuerza bruta*, ya que para cada observación hay N posibles estados para transitar y si tenemos $\{1, 2, ..., T\}$ observaciones tenemos

$N \times N \times ... \times N$, T veces por lo que la cantidad de operaciones es N^T y aproximadamente $2T$ productos en cada operación. Esta cantidad de operaciones no es factible de realizar, por ejemplo: para 10 estados ocultos y una simple secuencia de 100 símbolos habría que calcular $2 \times 100 \times (10^{100})$ operaciones; el número de átomos de hidrógeno en el universo se estima en cerca de 10^{80} átomos, por dar un ejemplo.

Para resolver el problema de la evaluación de manera eficiente se propone el procedimiento conocido como *procedimiento adelante-atrás* (del inglés *Forward-Backward procedure*) definiendo la variable *hacia adelante* $\alpha_t(i)$ como

$$\alpha_t(i) = P(O_1, O_2, ..., O_t, q_t = S_i \mid \lambda) \tag{4.6}$$

Y es posible encontrar una solución recurrente como sigue:

1. Inicialización:
$$\alpha_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \tag{4.7}$$

2. Inducción:
$$\alpha_{t+1}(j) = \left[\sum_{j=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}), \quad 1 \leq t \leq T-1,\ 1 \leq i \leq N \tag{4.8}$$

3. Terminación:
$$P(O \mid \lambda) = \sum_{t=1}^{N} \alpha_t(i) \tag{4.9}$$

Con el procedimiento anterior, en la ecuación 4.8 se observa que se necesitan N productos para N estados posibles y T posibles observaciones, con lo que la cantidad de operaciones es igual a N^2T. Para el caso de $N = 10$ estados ocultos y $T = 100$ símbolos emitidos la cantidad de operaciones es $10^2(100) = 100^3$ operaciones, que es razonablemente computable en un tiempo determinado.

La variable *hacia atrás* no es necesaria para resolver el problema de evaluación, pero resulta de utilidad para desarrollar el problema de aprendizaje de un modelo sobre las observaciones obtenidas. Se define $\beta_t(i)$ como

$$\beta_t(i) = P(O_1, O_2, ..., O_t \mid q_t = S_i, \lambda) \tag{4.10}$$

Y la solución recurrente se define como:

1. Inicialización:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{4.11}$$

2. Inducción:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \quad t = T - i,\ 1 \leq i \leq N \tag{4.12}$$

4.3.2. Estimar la secuencia óptima de estados

La ruta más probable en un modelo oculto de Markov es útil para el aprendizaje y para el alineamiento de secuencias con el modelo. La ruta más probable $Q = \{q_1, q_2, ..., q_T\}$ para la secuencia de observación $O = \{O_1, O_2, ..., O_T\}$ puede ser calculada utilizando el algoritmo de Viterbi [31].

Se define el valor

$$\delta_t(i) = \max_{q_1, q_2, ..., q_{t-1}} [P(q_1, q_2, ..., q_t = i, O_1, O_2, ..., O_t \mid \lambda)] \tag{4.13}$$

La ecuación anterior busca la mejor probabilidad para toda la observación O. Puede deducirse que se debe rastrear la mejor probabilidad en cada trayectoria posible en el tiempo t tomando en cuenta las trayectorias anteriores. Así se tiene que:

$$\delta_{t+1}(i) = \left[\max_i (\delta_t(i) a_{ij})\right] b_j(O_{t+1}) \tag{4.14}$$

El procedimiento completo se lista a continuación:

1. Inicialización:

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \tag{4.15a}$$

$$\psi_1(i) = 0 \tag{4.15b}$$

2. Recursión:

$$\delta_t(j) = \max_{1 \leq i \leq N} [(\delta_{t-1}(i) a_{ij}) b_{ij}], \quad 2 \leq t \leq N\ 1 \leq j \leq N \tag{4.16a}$$

$$\psi_t(j) = \arg\max_{1 \leq i \leq N} [\delta_{t-1} a_{ij}], \quad 2 \leq t \leq N\ 1 \leq j \leq N \tag{4.16b}$$

3. Teminación:

$$P^* = \max_{1\leq i\leq N}(\delta_T(i)) \tag{4.17a}$$

$$q_T^* = \arg\max_{1\leq i\leq N}(\delta_T(i)) \tag{4.17b}$$

4. Rastreo hacia atrás de la secuencia:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, ..., 1 \tag{4.18}$$

4.3.3. El problema de aprendizaje del modelo

El problema más difícil de los los modelos ocultos de Markov es determinar un método para ajustar los parámetros A, B y Π del modelo para satisfacer los criterios de optimización. No se conoce una forma analítica para fijar los parámetros que maximicen la probabilidad de la secuencia de observación. Existen varios algoritmos disponibles para el entrenamiento de un modelo oculto de Markov, entre ellos, el algoritmo Baum-Welch que es un método iterativo [31].

Se define la variable $\xi_t(i,j)$ como:

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j \mid O, \lambda) \tag{4.19}$$

De las definiciones de $\alpha_t(i)$ y $\beta_t(i)$ en las ecuaciones 4.6 y 4.10 se tiene la probabilidad condicional

$$\xi_t(i,j) = \frac{P(q_t = S_i, q_{t+1} = S_j \mid O, \lambda)}{P(O \mid \lambda)} = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P(O \mid \lambda)} = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)} \tag{4.20}$$

Se define $\gamma_t(i)$

$$\gamma_t(i) = P(q_t = S_i \mid O, \lambda) \tag{4.21}$$

Y nuevamente se expresa en términos de las variables $\alpha_t(i)$ y $\beta_t(i)$ como

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)} \tag{4.22}$$

Y $\gamma_t(i)$ se relaciona con $\xi_t(i)$ de la forma

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{4.23}$$

La suma en la ecuación 4.23 puede interpretarse como el número esperado de veces que se pasa por el estado S_i o el número de transiciones de S_i dada O, excluyendo el tiempo $t = T$. Similarmente la suma en la ecuación 4.20, desde $t = 1$ hasta $t = T - 1$ se interpreta como el número esperado de transiciones del estado S_i al estado S_j.

De forma breve tenemos:

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{Número esperado de transiciones por } S_i \text{ en } O \tag{4.24}$$

$$\sum_{t=1}^{T-1} \xi_t(i) = \text{Número esperado de transiciones de } S_i \text{ a } S_j \tag{4.25}$$

Así, el procedimiento para actualizar los parámetros de un modelo oculto de Markov se realiza simplemente contando la ocurrencia de transiciones. Lo que genera el procedimiento de estimación mostrado a continuación:

$$\hat{\pi}_i = \text{Frecuencia en el estado } S_i \text{ en } t \text{ igual a } 1 = \gamma_1(i) \tag{4.26}$$

$$\hat{a}_{ij} = \frac{\text{Número de transiciones de } S_i \text{ a } S_j}{\text{Número esperado de transiciones por } S_i} = \frac{\sum_{t=1}^{T-1} \xi_t(i)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{4.27}$$

$$\hat{b}_j(k) = \frac{\text{Número de veces en } S_j \text{ observando } v_k}{\text{Número de veces en } S_j} = \frac{\sum_{Num.O_t=v_k}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \tag{4.28}$$

Existen adicionalmente tres problemas técnicos en la implementación en un modelo oculto de Markov, todos estos se tratan a detalle en [31] con su respectiva solución. Los tres problemas son:

- El Reescalamiento de las probabilidades en cada iteración durante la ejecución de los algoritmos.
- El tratamiento de observaciones con más de un solo símbolo (vectores de símbolos).
- Inestabilidad en la representación númerica. Al multiplicar recursivamente valores en el intervalo $[0, 1]$.

Finalmente, con la estimación de la secuencia óptima de estados y la evaluación de una secuencia, el problema de estimar un modelo queda resuelto, ya que se pueden optimizar los parámetros para obtener la mayor probabilidad dada la secuencia actual de símbolos, comparada con su transición de estados óptima. Con esto, es posible entrenar modelos probabilísticos para representar clases de imágenes con contenido visual similar. Por lo que se puede obtener una memoria visual de un ambiente. Para ello, se deben entrenar los modelos correspondientes y consultarlos, como por ejemplo: construir un grafo de imágenes en base a su similitud visual. Si dos imágenes son semejantes se pueden agregar en un grafo como dos aristas unidas por un vertice.

Con lo anterior es posible crear una memoria auxiliar para mejorar el desempeño de un sistema de aprendizaje por refuerzo utilizando solo información visual. Esta idea tambien se plantea en el siguiente capítulo utilizando hash sensible a la localidad (HSL).

Capítulo 5

Hash Sensible a la Localidad

En este capítulo se describe el segundo método probabilístico para poder indexar imágenes similares en el ambiente. El objetivo de este es poder crear una representación que funcione como una memoria auxiliar. Para este caso se trata de un grafo formado con imágenes. Este grafo funciona como una memoria visual para agilizar la convergencia del aprendizaje reforzado, como mecanismo de atención en el ambiente facilitando la convergencia en el entrenamiento.

La sección 5.1 describe cómo es posible utilizar funciones hash para indexar imágenes, respetando la similitud entre ellas. Para esto se construye un modelo probabilístico basado en similitud de descriptores binarios y este se usa como criterio para agregar vértices a un grafo.

5.1. Agrupamiento con hash sensible a la localidad

Una *tabla hash* o una tabla fragmentada es una estructura de datos que asocia valores de entrada a un índice. Después índice es usado para acceder al valor que se desea consultar en esta estructura. Este mecanismo es denomindado acceso *clave-valor* y suele utilizarse en sistemas de búsqueda de datos que tienen una especificación de tiempo de acceso muy eficiente. Típicamente, el tiempo de acceso es diseñado para ser de tiempo constante $O(1)$ [32].

El *hash sensible a la localidad* (HSL) es una técnica que aplica el hash de elementos de entrada similares en los mismos bloques de una tabla hash. El hash sensible a la localidad es una técnica de indexación que permite buscar de manera eficiente los vecinos más cercanos entre grandes colecciones de elementos, donde cada elemento está representado por un vector de entrada $x^d = \{x_1, x_2, ..., x_d\}$, para algún número de dimensiones d. El algo-

ritmo es aproximado, pero ofrece garantías probabilísticas. Es decir, con la configuración de parámetros correcta, los resultados aproximan a realizar una búsqueda de fuerza bruta en toda la colección de datos pero con un tiempo de consulta más eficiente [33].

Sea un vector de entrada $x^d \in d$, y una función de hash $h(\ )$, la salida i se define como:

$$i = h(x^d) \tag{5.1}$$

En un sentido general se busca conseguir un tiempo de acceso constante al índice i o a una dirección de memoria, pero puede diseñarse una función h que maximize las colisiones en las entradas de la tabla en lugar de minimizarlas. La figura 5.1 ilustra esta idea.

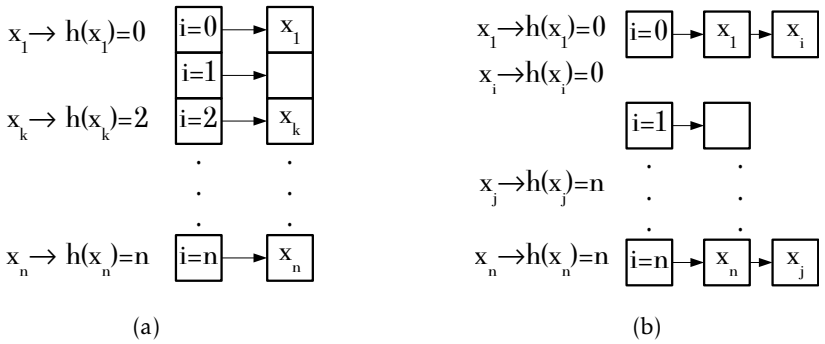


Figura 5.1: Diferencia entre hash sensible a la localidad y hash común. a) Hash para acceso eficiente. b) Hash sensible a similitud o localidad.

El hash sensible a la localidad se basa en el concepto de similitud entre dos conjuntos. Dados A y B dos conjuntos, una medida de similitud común de usar es la denominada *similitud de Jaccard*, que se define como:

$$J(A,B) = \frac{|\ A \cap B\ |}{|\ A \cup B\ |} \tag{5.2}$$

Por ejemplo, sean los conjuntos $A = \{1,2,3\}$ y $B = \{3\}$. La similitud de Jaccard es $J(A,B) = \frac{|\{3\}|}{|\{1,2,3\}|} = \frac{1}{3}$.

Igualmente podemos representar conjuntos de números binarios con el propósito de explotar las operaciones binarias de una computadora. Un ejemplo es $A_b = \{1,1,0\}$, $B_b = \{0,1,0\}$ y $C_b = \{1,1,1\}$. Tenemos que $J(A_b,B_b)$ se define como:

$$J(A_b,B_b) = \frac{M_{11}}{M_{10} + M_{01} + M_{11}} \tag{5.3}$$

Donde M_{11} es la cantidad de símbolos '1' en ambos conjuntos. Para el valor M_{10} es la cantidad de símbolos '1' en A y '0' en B. Finalmente, M_{01} es la cantidad de símbolos '0' en A y '1' en B.

Por lo que $J(A_b, B_b) = \frac{1}{1+0+1} = \frac{1}{2}$ y $J(A_b, C_b) = \frac{2}{0+1+2} = \frac{2}{3}$.

También puede establecerse una métrica en un espacio de vectores. Esta se define como:

$$J_D(S, A) = 1 - \frac{| A \cap B |}{| A \cup B |} \tag{5.4}$$

Y para el caso de vectores binarios tenemos:

$$J_{D_b}(A_b, B_b) = 1 - \frac{M_{11}}{M_{10} + M_{01} + M_{11}} \tag{5.5}$$

Ahora supongamos que A_b, B_b y C_c son una permutación aleatoria de algún conjunto binario con tres elementos. Entonces se define a la función $h_{min}(\ )$ como la primera posición donde ocurre el primer valor '1'. Formalmente, se define una permutación de los elementos de un conjunto S a cualquier otro conjunto distinto:

$$h : S -> \mathbb{N} \tag{5.6}$$

Entonces h_{min} define una función hash h donde se regresa el valor hash menor de los elementos de S:

$$h_{min}(S) = min(\{h(x) \mid x \in S)\}) \tag{5.7}$$

Por ejemplo: $h_{min}(A_b) = 1$, $h_{min}(B_b) = 2$ y $h_{min}(C_b) = 1$.

Se ha demostrado en [34] que se puede establecer la relación para comparar la similitud en conjuntos de la forma:

$$p[h_{min}(A) = h_{min}(B)] = J(A, B) \tag{5.8}$$

Donde la probabilidad $p[h_{min}(A) = h_{min}(B)]$ es establecida por la similitud entre los valores hash de A y B.

La manera de describir un conjunto $H = \{h_1, h_2, ..., h_k\}$ de funciones hash, a las que se dice son $(R, cR, p_1, p_2) - sensibles$ respecto a una función de similitud o distancia $d(x, y)$, es que si para cada $h \in H$, se cumple que:

$$Si\ d(x, y) \leq R,\ \ entonces\ \ p_H[h(x) = h(y)] \geq p_1 \tag{5.9}$$

$$Si\ d(x,y) \geq cR,\ \ entonces\ \ p_H[h(x) = h(y)] \leq p_2 \tag{5.10}$$

Las probabilidades p_1 y p_2 son conocidas como probabilidades de colisión para dos elementos de entrada x e y. La figura 5.2 ilustra esta idea.

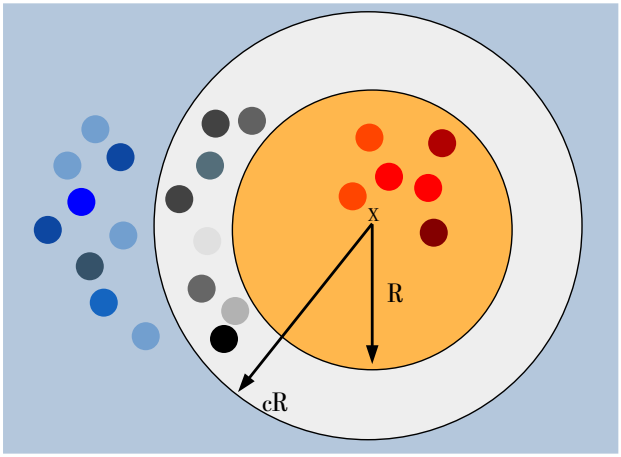


Figura 5.2: Similitud entre elementos con funciones H, $(R, cR, p_1, p_2) - sensibles$.

Por lo que una tabla hash sensible a la localidad (HSL), usada para clasificar vectores tendrá la estructura mostrada en la figura 5.3. La figura ilustra las colisiones para elementos semejantes en base a la clase de los elementos. Esto se representa como el color, y cada bloque de la tabla solo contiene elementos semejantes según la métrica $d(x,y)$ o con probabilidad semejante según su código hash.

El tiempo de consulta de una tabla HSL es comúnmente $O(mkd + búsqueda\ lineal)$ donde m es un número de muestreo para establecer la relación $h_i(x) = h_i(y),\ \ \forall i \in m$. A veces denominado mapeo por *m-grama* para encontrar funciones hash adecuadas, donde $k \leq d$ [35]. El número k es la cantidad de funciones $\{h_i\}$. Y el valor d es la dimensión de los datos a almacenar. Una vez obtenido el código hash, solo se hace una búsqueda lineal en ese bloque.

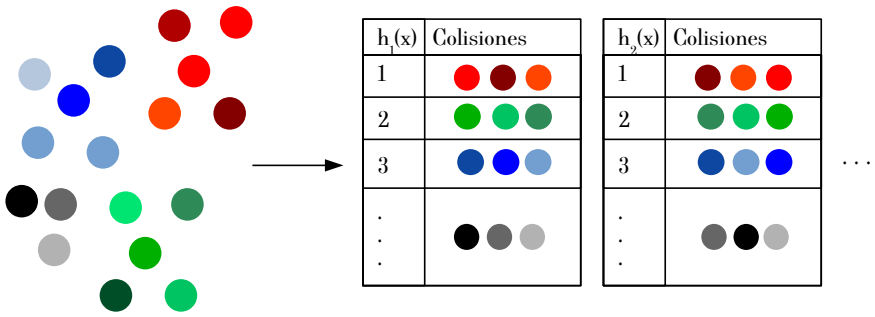


Figura 5.3: Tabla hash sensible a la localidad o similitud. Las colisiones se muestran según el color o la clase de los elementos.

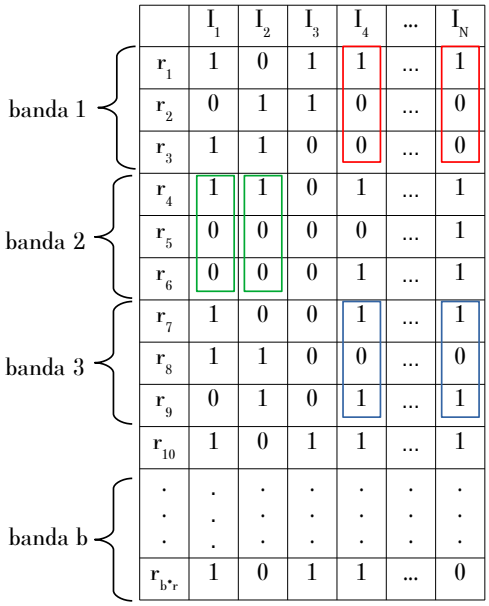
Por lo anterior, puede usarse hash sensible a la localidad con funciones $h \in H$ que cumplan la propiedad en la ecuación 5.8, para comparar imágenes digitales previamente transformadas en secuencias binarias. Lo que genera colisiones en los bloques de la tabla, según su similitud.

Un método para aplicar tablas HSL es el denominado *partición de b bandas*. Este método consiste en particionar los descriptores binarios que representan cada imágen en bloques de renglones para su posterior asignación. Primero se convierten los descriptores binarios en una matriz formando un arreglo de $(b \times r) \times N$ descriptores. Con cada descriptor I_i, de tamaño $(b \times r) \times 1$. La figura 5.4 ilustra este procedimiento para valores de $r = 3$.

Los rectangulos de colores indican la colisión por banda, para imágenes distintas. En la figura se aprecia en color rojo una colisión para I_4 e I_N en la banda 1 En color verde una colisión para I_1 e I_2 en la banda 2. Y en color azul una colisión para I_4 e I_N en la banda 1. Entre más bandas sean indexadas al mismo bloque de cada tabla HSL la similitud de Jaccard será mayor.

Entonces podemos estimar la probabilidad de que la firma de una banda, tenga el mismo valor entre imágenes distintas. Para esto definimos lo siguiente:

- Sea $p_1 = p^r$ la probabilidad p de que los renglones sean iguales entre imágenes en una sola banda de tamaño r.
- Sea $p_2 = 1 - p^r$ la probabilidad p de que los renglones no sean iguales entre imágenes en una sola banda.
- Sea $p_3 = (1 - p^r)^b$ la probabilidad p de que ningún renglón en cada banda, sea igual para todas las bandas b.



	I_1	I_2	I_3	I_4	...	I_N
r_1	1	0	1	1	...	1
r_2	0	1	1	0	...	0
r_3	1	1	0	0	...	0
r_4	1	1	0	1	...	1
r_5	0	0	0	0	...	1
r_6	0	0	0	1	...	1
r_7	1	0	0	1	...	1
r_8	1	1	0	0	...	0
r_9	0	1	0	1	...	1
r_{10}	1	0	1	1	...	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .
r_{b*r}	1	0	1	1	...	0

Figura 5.4: Ejemplo de construcción de bandas.

- Se define la probabilidad de que al menos una banda de todas, para todas las imágenes I_i, $\forall i \in N$ coincida entre dos imágenes distintas como:

$$P = 1 - [(1 - p^r)^b] \tag{5.11}$$

Donde p es la similitud de Jaccard entre los códigos binarios de las imágenes:

$$p = [h(I_i) = h(I_j)] = J(I_i, I_j)$$

La figura 5.5 muestra un ejemplo de la relación entre la probabilidad de colisión P entre firmas de imágenes divididas en bandas y la similitud de Jacccard. Esta similitud está parametriza en p, que es el umbral de decisión que podemos elegir. Dependiendo del número de bits en las bandas y el número de ellas, es posible aumentar o disminuir la precisión de la similitud entre imágenes. La figura 5.5 muestra la relación $P[J(I_i, I_j); r, b]$ contra $J(I_i, I_j)$ para $r = 8$ y $b = 32$ formando 256 renglones binarios. Para un umbral $p \geq 0.7$ es posbile garantizar una probabilidad de similitud $P \geq 0.85$. Para un umbral $p \geq 0.75$, es posible garantizar una probabilidad de

similitud $P \geq 0.95$.

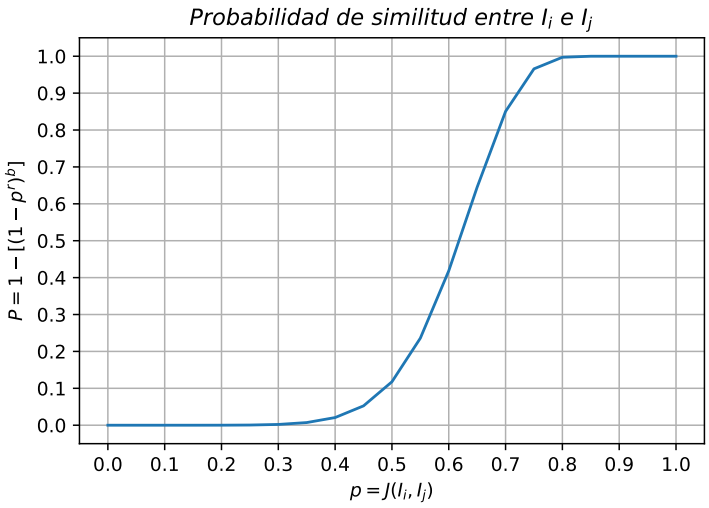


Figura 5.5: Relación P y $J(I_i, I_j)$ para $r = 8$ y $b = 32$.

En el siguiente capítulo se describe el propósito de utilizar una representación de grafo para las imágenes del ambiente, con el fin de brindar información topológica visual al sistema de navegación entrenado con aprendizaje reforzado. Este grafo es construido uttilizando métodos estadísticos para representar las imágenes y su similitud.

Capítulo 6

Memoria Visual por Similitud Estadística

En este capítulo se describe el uso de una memoría visual como subsistema del sistema de aprendizaje por refuerzo, está memoria tiene el objetivo de proveer información topológica en forma de una abstracción visual para mejorar el entrenamiento. En la sección 6.1, se presenta una introducción al problema de buscar imágenes semejantes en un base de datos. En la sección 6.2 se expone el como utilizar modelos ocultos de Markov (MOM) como herramienta para buscar similitudes entre imágenes. En la sección 6.3 se aborda el mismo problema pero utilizando hash sensible a la localidad (HSL). Finalmente, la sección 6.4 explica como se construye la memoria visual.

6.1. Memoria visual por similitud estadística

En la actualidad, con el gran aumento de dispositivos móviles de comunicación, su conexión a internet y su bajo costo. Las bases de datos de imágenes requieren de mecanismos eficientes de consulta y actualización, por lo que se ha incrementado la investigación en el area de *recuperación de imágenes basada en contenido* [36]. En esta sección se describe como realizar esta tarea con el objetivo de mejorar el desempeño de un sistema basado en aprendizaje por refuerzo, ya que este requiere de la exploración del ambiente de manera extensiva.

Con el objetivo de buscar el contenido visual de una imagen en una base de datos de forma rápida, robusta y eficiente, es necesario examinar el contenido en cada imagen para transformarlo en una representación que sea útil en presencia de cambios de escala, iluminación y rotación. Para lograr esto, es necesario asignar una firma o descriptor a

cada imágen y después establecer una estructura de datos que nos permita realizar una consulta.

La estructura general de un sistema de búsqueda por contenido visual se muestra en la figura 6.1. Dado un conjunto de datos construido previamente y almacenado, se consulta una imagen aplicándole una etapa de preprocesamiento, extracción y descripción de su contenido. Los descriptores son ultilizados para representar a la imagen en un dominio predeterminado. Este dominio puede ser un espacio vectorial continuo o discreto. Al final, el sistema establece un criterio de similitud y de búsqueda para dar como salida los índices de las imágenes más semejantes. Si la base de datos no es estática, es posible reevaluar su estructura para mejorar su consulta a través de un módulo de evaluación de las salidas.

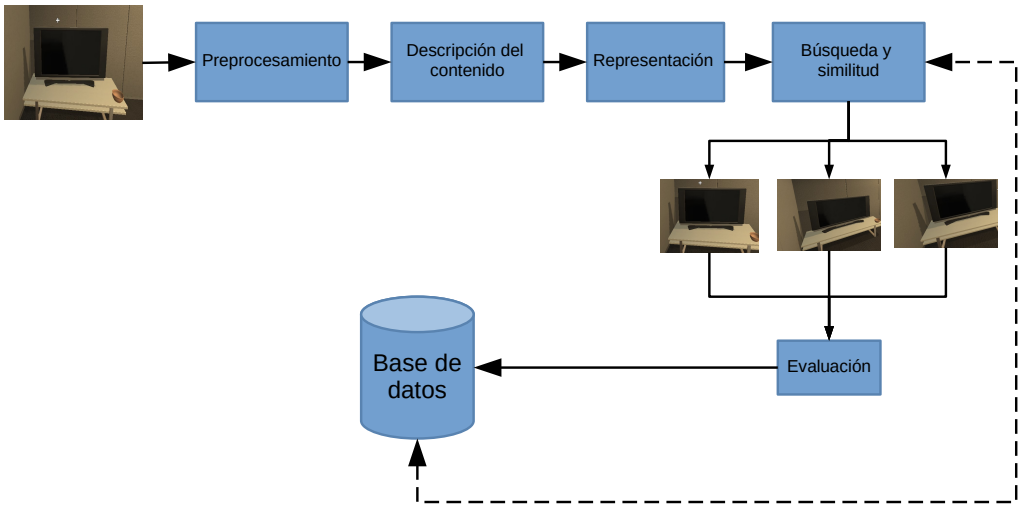


Figura 6.1: Diagrama de bloques de un sistema de recuperación de imágenes.

Para obtener una firma única de cada imagen, y la posterior construcción de una base de datos con dichas representaciones, se suele establecerse una jerarquía en el contenido visual, en tres niveles: contenido de bajo nivel; contenido de nivel medio; y contenido de alto nivel.

- Contenido de bajo nivel. Son las llamadas características primitivas de la imagen como son el color, textura, forma y posición.

- Contenido de nivel medio. Son características obtenidas de grupos de píxeles como son los objetos, contornos o bordes.

- Contenido de alto nivel. Suelen ser construcciones con el uso de detectores de alto nivel como detección de escenas, rostros o emociones.

Conforme más alto es el nivel del contenido visual en la lista anterior, tienden a aumentar los requerimentos de cómputo y de latencia para su construcción (ver figura 6.2).

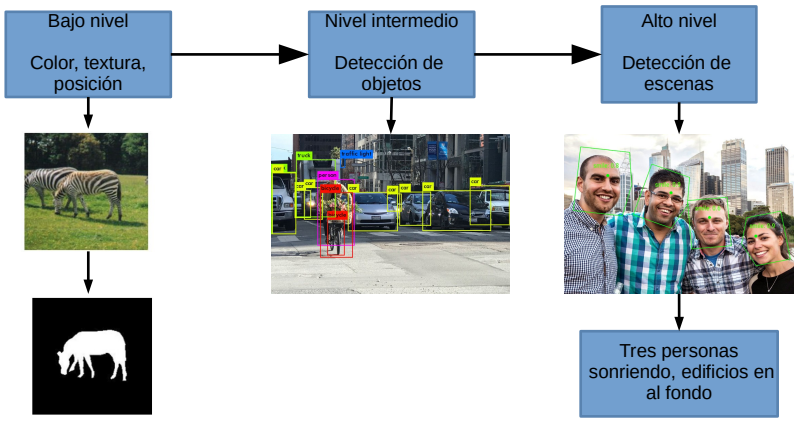


Figura 6.2: Escalas de contenido visual en una imagen [37] (imágenes procesadas con [38]).

Para representar el contenido es común dar una representación de tipo vectorial, ya sea de manera global o local. Para una representación global se construye un solo vector para cada imagen. Para el caso local, se suele dividir la imagen en secciones conservando alguna relación local o de grupo entre los pixeles. Los vectores pueden contener entradas con valores discretos o valores continuos, típicamente los valores discretos son más compactos en su representación y tienen mayor velocidad de lectura.

Algunos métodos de clasificación de los descriptores son las máquinas de soporte de vectores, n-vecinos más cercanos, técnicas de agrupamiento no supervisado o técnicas de acceso y búsqueda con árboles. Algunos métodos actuales se revisan en [39].

En el caso de este trabajo de investigación, se ha seleccionado la descripción de bajo nivel para las imágenes debido a la velocidad para buscar en miles de imágenes y poder generar un grafo topológico. El objetivo es el de mejorar el aprendizaje por refuerzo en un sistema de navegación visual, esto se discute más a detalle en la sección 7.

6.2. Memoria topológica comparando modelos ocultos de Markov

El capítulo 4 presentó la teoría matemática de los modelos ocultos de Markov (MOM). Esta teoría permite utilizar un modelo probabilístico gráfico para detectar secuencias. Dichas secuencias pueden ser vectores, símbolos, series de tiempo e incluso secuencias de imágenes. En esta sección se describe cómo se utilizar este método para buscar probabilísticamente el contenido visual en imágenes capturadas en un entorno virtual, utilizado por un robot simulado.

Para poder utilizar un MOM como detector probabilístico de imágenes, es necesario construir secuencias de símbolos. En este trabajo de investigación, los símbolos son vectores obtenidos a partir de construir secciones uniformes sobre las imágenes de entrenamiento. La figura 6.3 muestra cómo se hacen estas regiones. Cada region es transformada al espacio de color *CIE Lab* y se forman vectores de dimensión $3 \times (n \times m)$ donde n y m son las dimensiones del segmento y el número 3 corresponde a los canales del espacio de color CIE L, a, b.

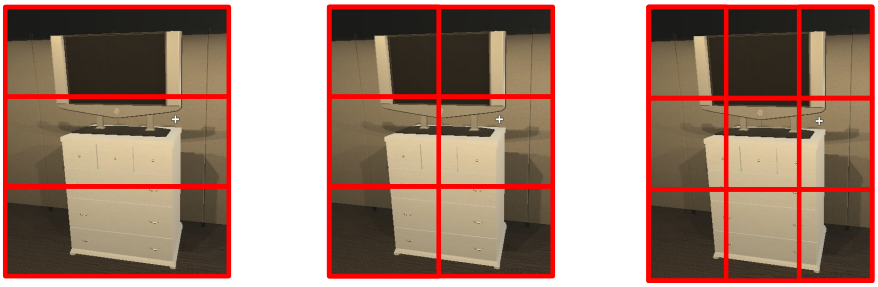


Figura 6.3: Regiones para construir símbolos [40]. Cada rectángulo forma un vector. Se contruyen modelos para 3 estados, 6 estados y 9 estados.

Para el entrenamiento se usan tres modelos paralelos para 3, 6 y 9 estados. Esta idea fue publicada en *Comparison of two objects classification techniques using hidden markov models and convolutional neural networks* [40].

Las imágenes son capturadas cada 5 grados en un punto en el ambiente simulado. El espacio es seleccionado cuantizando el espacio navegable en el simulador usando k-medias. Esto se puede apreciar para una escena simulada, en la figura 6.4 donde se marca con circulos rojos las coordenadas de la captura de las imágenes.

Cada MOM se contruye representando la imagen que se capturó a 15 grados de rotación, a partir de 0 grados en el sistema de referencia del simulador en la coordenada

Figura 6.4: Los puntos rojos muestran en donde se ha capturado el contenido visual de una escena de ejemplo.

seleccionada en el proceso de cuantizar el espacio libre.

Posteriormente, se construye un grafo. En este grafo las aristas representan las conexiones con las cinco imágenes más parecidas respecto una imágen tomada cada 15 grados. Se evalúa cada modelo para saber cuales son las 5 imágenes más semejantes y en caso de serlo, se conectan al vertice. La detección es ejecutada como se muestra en la figura 6.5. Se compara cada MOM que representa las imágenes tomadas a 15 grados. El modelo más probable será la imagen más parecida a la que se consulta.

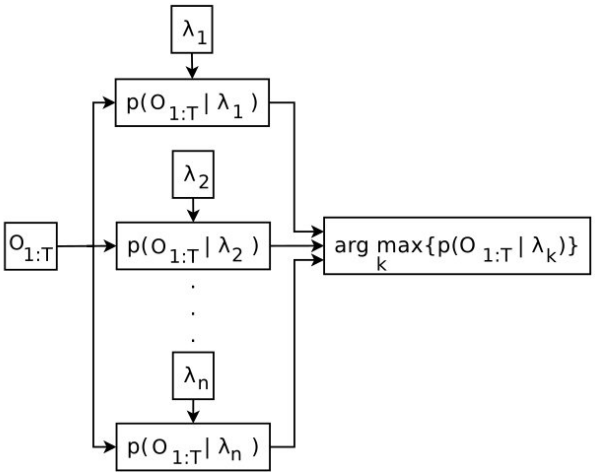


Figura 6.5: Sistema de detección con MOM.

Al final, se obtiene un grafo de similitud visual que se usará como información adicional para el entrenamiento del sistema basado en aprendizaje por refuerzo.

6.3. Memoria topológica usando hash sensible a la localidad

La sección 6.3 expusó la idea para utilizar hash sensible a la localidad (HSL). El procedimiento utilizado en este trabajo para construir la memoria visual es el siguiente:

Primeramente, se selecciona un tamaño de firma binaria. Típicamente k^2, en donde k debe ser potencia de 2 para explotar la simetría en los registros en el CPU de una computadora, los cuales suelen ser de 32 bits para operaciones de precisión simple. De 64 bits para instrucciones de precisión doble, o de 256 bits en bibliotecas compiladas para instrucciones denominadas *AVX2*. Este es el caso de este trabajo en el que se usa la bilbioteca *NumPy* [41] y *FALCONN* [33] entre otras. Además, k está relacionada con el número de bandas, esto es $k^2 = b \times r$.

Después se convierte cada imagen a escala de grises, por lo que se tiene una imagen de intensidad, con tamaño $(k+1) \times k$ píxeles como se ve en la figura 6.6. Se crea una firma binaria comparando el valor de cada píxel con el siguiente. De arriba abajo y de izquierda a derecha. Si el valor del pixel siguiente es menor que el anterior, se asigna el símbolo '0', si es mayor o igual se asigna '1'.

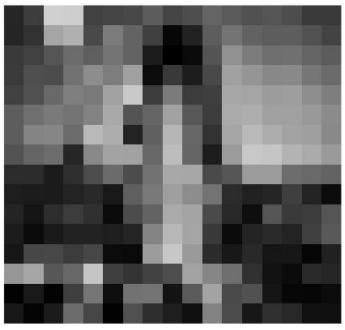


Figura 6.6: Conversión a escala de grises. Tamaño de 17×16, para $k = 16$.

La figura 6.7 muestra un ejemplo de firma binaria.

Por último, se construyen tablas HSL de búsqueda usando los códigos binarios y el método de hashing sensible a localidad, cada tabla aloja a cada banda.

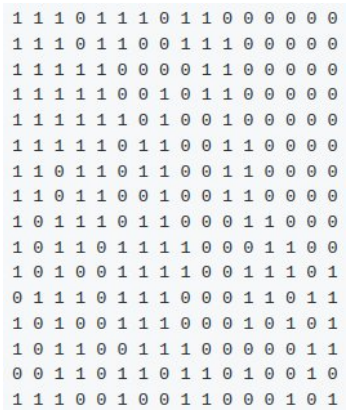


Figura 6.7: Conversión a código binario. $k^2 = 256$ bits.

6.4. Grafo topológico por similitud visual

En las dos secciones anteriores se mencionó el procedimiento para identificar correspondecias por similitud visual entre imágenes. El objetivo de esto es el formar una representación visual del entorno para mejorar un sistema de navegación por aprendizaje reforzado. Esta memoria se construye formando un grafo $G(V, A)$, donde el conjunto $\{V\}$ representa los vértices y el conjunto $\{A\}$ las aristas en cada vertice. Para una imagen consultada I_i, se construye una función F que regresa como salida una arista del grafo G, es decir:

$$a_i = F[I_i, G(V, A)], \ \forall i \in \{A\} \tag{6.1}$$

El grafo G se construye utilizando todos los MOM construidos para algunos puntos muestreados en el ambiente, y también utilizando tablas HSL generando dos casos distintos:

1. Para el caso de los MOM. Se toma una imagen de consulta I_i, se crean secuencias de vectores requeridos por el método y se consultan los 5 modelos λ_k más probables (se ensamblan 3 modelos por imágen para 3, 6 y 9 estados).
   Posteriormente, se genera un diccionario para formar el grafo y almacenar en cada identificador de arista un descriptor de la imágen y sus 5 vertices asignados a las imágenes más probables. La figura 6.8 muestra un ejemplo de como se conectan los vertices capturando la relación visual entre imagenes.

2. Para el caso de la tabla HSL. Se toma una imagen de consulta I_i, se crean una secuencia binaria requerida por el método y se consultan las 5 imágenes con mayor

similitud visual.
Posteriormente, se genera un diccionario para formar el grafo y almacenar en cada identificador de arista un descriptor de la imágen, su firma binaria y sus 5 vertices asignados a las imágenes más probables.

En el siguiente capítulo se detalla que el descriptor utilizado es algunos de los siguientes: la firma binaria para HSL de tamaño 1×256 enteros; un descriptor ResNet-18 de tamaño 1×512 flotantes; o un descriptor ResNet-50 de tamaño 1×2048 flotantes.

3. Finalmente, teniendo el grafo $G(V,A)$ para los dos métodos se construye un nuevo diccionario, para poder localizar una imagen de consulta con la arista más parecida y el descriptor contenido en esa posición. Esto con el fin de extraer el descriptor para alimentar un sistema de aprendizaje reforzado.

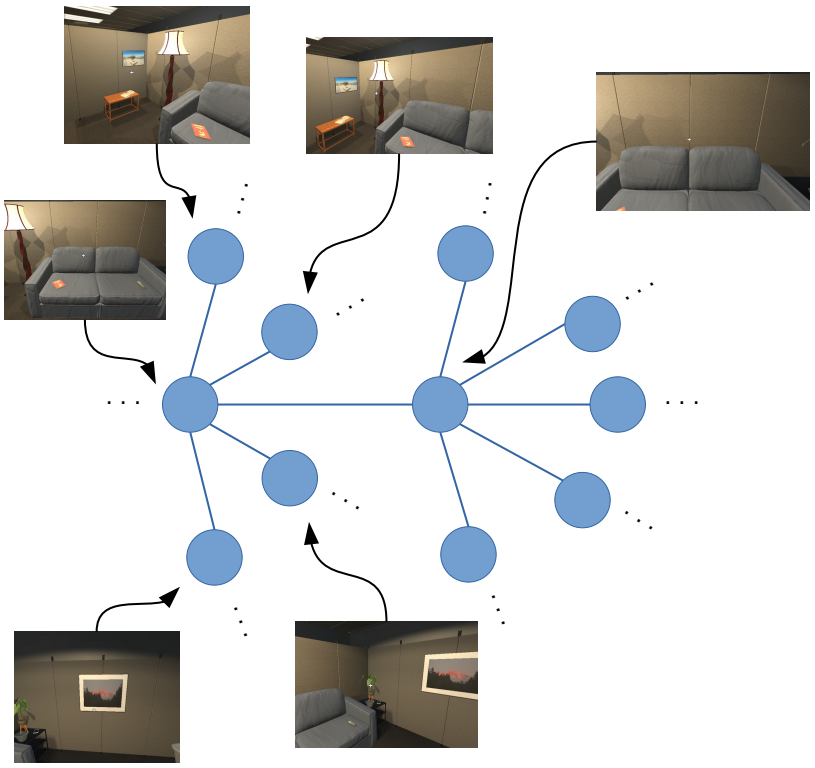


Figura 6.8: Grafo de similitud visual.

Capítulo 7

Navegación Visual Usando Memoria Topológica y Objetivos

En esta sección se describe el sistema de navegación visual planteado en este trabajo de investigación. La idea principal se basa en utilizar únicamente imágenes junto con una memoría auxiliar del ambiente, contruida a partir de ellas, para aprender comportamientos de navegación en espacios interiores. Con este propósito, se plantea un sistema de navegación condicionado a un objetivo en el ambiente. La intención de usar este enfoque, es aumentar la información que se introduce en el sistema mejorando el aprendizaje de una política de acción.

La sección 7.1 contiene un resumen de la arquitectura propuesta. La sección 7.2 describe las herramientas de simulación de ambientes interiores utilizada.

7.1. Arquitectura

La principal aportación de este trabajo de investigación es la incorporación de una memoria auxiliar, suministrada al sistema en forma de grafo. Esta memoria tiene el objetivo de acelerar la convergencia del sistema basado en aprendizaje por refuerzo, suministrando dos tipos de objetivos visuales.

- El primer tipo de objetivo es una imágen del destino en el ambiente, este condiciona la tarea de navegación de manera global hacia algún lugar.

- El segundo tipo, es un objetivo visual local dado el estado actual en el ambiente.

En el caso del sistema de navegación, es entrenado utilizando aprendizaje por refuerzo. De forma que, al explorar ambientes simulados foto realistas el programa explora la ejecución de acciones de navegación. Esto puede ser de manera secuencial o utilizando la paralelización para una misma política de acción. Esto posibilita el entrenamiento mediante experiencia, en presencia de errores creados artificialmente como son de desplazamiento y ruido en las imágenes, explotando la capacidad de poder extraer cientos de imágenes por segundo y por cada instancia del sistema. Esto es un práctica común en sistemas de aprendizaje profundo en conjunto con aprendizaje por refuerzo [42] en conjunto con aprendizaje profundo.

La figura 7.1 ilustra el sistema de navegación de manera general. La imagen capturada en algun tiempo t de exploración del ambiente y la imagen objetivo.

Se extraen características visuales mediante la inferencia de una red neuronal profunda denomida *ResNet* [17], a la cual se le han removido las capas de clasificación. La diferencia de las características visuales del estado y del objetivo encontradas en la memoria, se normalizan y se concatenan para formar un vector que contiene información del estado actual y del objetivo visual. Finalmente, con los vectores concatenados, se alimenta un sistema de aprendizaje por refuerzo el cual es simplemente un perceptrón multicapa y con unidades recurrentes.

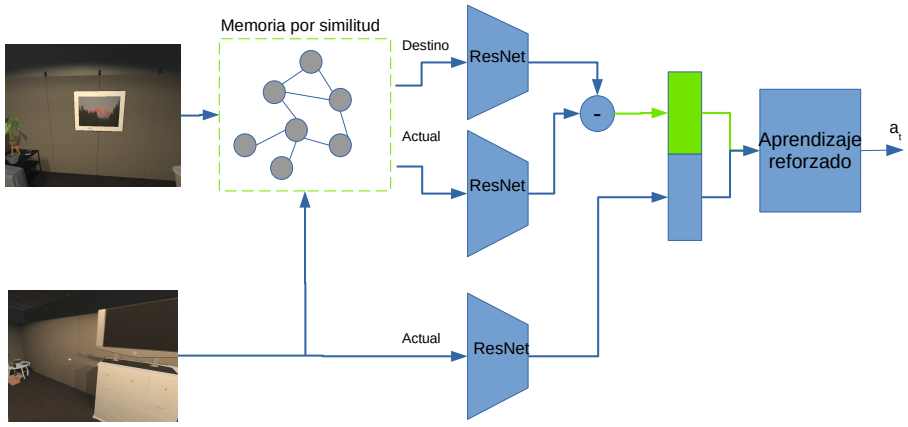


Figura 7.1: Diagrama de bloques del sistema propuesto.

La arquitectura en detalle se muestra en la figura 7.2. El sistema se compone de tres bloques principales: un bloque de extracción de características y representación del ambiente; una memoria topológica; y un sistema de aprendizaje por refuerzo con observaciones locales y un objetivo visual global.

- Extracción de objetivos y características. En este módulo se obtiene el objetivo visual al que se navegará. Esto tiene el objetivo de reducir la complejidad de la tarea de navegación, para eso se construyó una memoria topológica de manera previa en el ambiente. Se utiliza la red neuronal profunda ResNet-18 como base (esta red está preentrenada en la base de datos *ImageNet*[43]).

  Con las características visuales obtenidas, estas se concatenan para describir de manera conjunta el lugar a donde llegar y el estado actual. También se calcula la diferencia entre las características que coinciden en la memoria, se normalizan y se usan para poder representar que tan lejos se encuentra el robot el objetivo final conforme se desplaza.

- Memoria por apariencia visual. Este módulo se construye utilizando los métodos estadísticos descritos en el capítulo 6. Se utilizan los modelos ocultos de Markov para representar cada imagen y poder determinar cuál es la imagen más probable que esté en la memoria, respecto al estado actual. De manera semejante se usa hash sensible la localidad.

  Con lo anterior es posible construir un grafo de imágenes únicamente utilizando la apariencia visual de bajo nivel. Esto es, utilizando los píxeles para formar un descriptor y almacenarlo en una base de datos junto con un un grafo que representa el ambiente.

- Sistema de ejecución de acciones. Este módulo utiliza aprendizaje por refuerzo, que es un mecanismo de aprendizaje de una función para ejecutar acciones en base a las observaciones actuales. La parte mostrada en líneas punteadas azules en la 7.2 es una red neuronal, que utiliza las características visuales para optimizar una política de acción que da a su salida acciones asociadas a la rotación, al avance o al detener la ejecución.

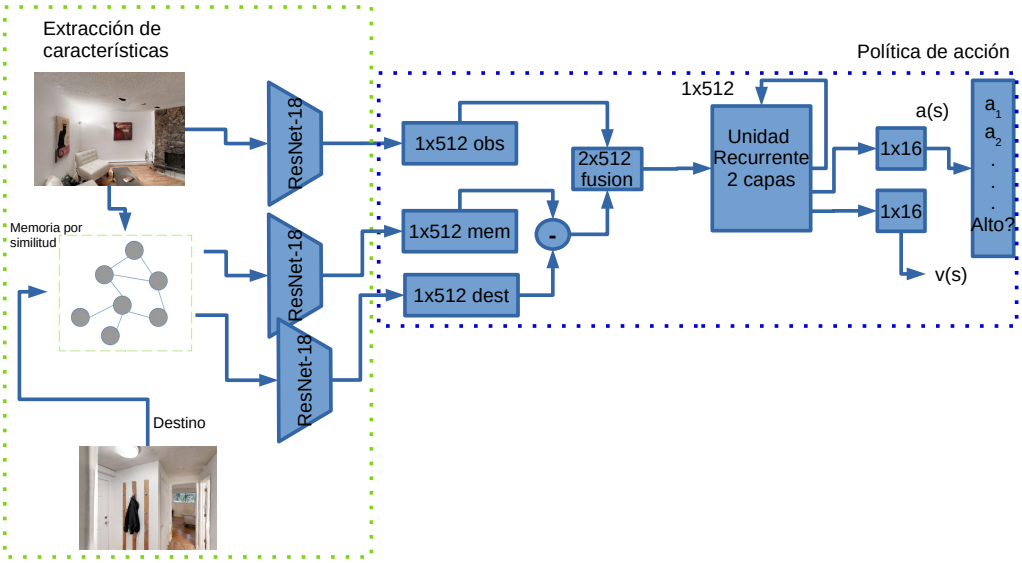


Figura 7.2: Detalle de la arquitectura propuesta.

7.2. Herramientas de desarrollo y simulación fotorrealista

Actualmente, se ha vuelto un método común simular el comportamiento de un agente inteligente usando software de simulación para videojuegos como es el entorno *Unity* o *Unreal Engine*. Unreal Engine fue desarrollado desde la década de 1990 en lenguaje C y C++ con el objetivo de desarrollar videojuegos en 3D en con física realista. Para el caso de Unity, este fue desarrollado en el año 2000 por los creadores de OpenGL, y ahora es el motor de videojuegos más utilizado. Por esto, estos entornos de desarrollo cuentan con una física muy optimizada debido a su éxito comercial. En el caso de Unity, provee una interfaz de desarrollo en lenguaje C# actualmente. Por otra parte, Unreal Engine está completamente desarrollado en C++ y las bibliotecas están suministradas en el mismo lenguaje. Finalmente, debido al auge del lenguaje python, ahora existen motores de física como *PyBullet* creados en lenguaje C++, pero encapsulados usando la biblioteca *Cython* para brindar una interfaz más ágil de desarrollo.

A partir del año 2020, la mayoría de las grandes universidades han comenzado a desarrollar sistemas de simulación robustos para facilitar el desarrollo de agentes con cuerpos mecatrónicos. Se pretende evitar problemas en la transferencia de aprendizaje a sistemas robóticos y facilitar el desarrollo de sistemas inteligentes. Algunas empresas y universidades de Estados Unidos que encabezan esta tendencia son *Google*, *Facebook*, *Stanford*, *MIT*, *AI2* (del inglés *Allen Institute for Artifical Inteligence*) entre otros.

El instituto Allen, Standford y Facebook proveen los simuladores más desarrollados los cuales respectivamente son *AI2-THOR* [20], *iGibson* [44] y *Habitat*[45]. AI2-THOR y Habitat ya cuentan con sus propias bibliotecas de software para agentes autónomos, además de las funciones de operación de los simuladores. iGibson aún depende de paquetes y versiones específicas de ROS. Todos incorporan cinemática y dinámica para algunos robots como *Roomba*, *Fetch*, *Locobot* entre otros.

AI2-THOR está equipado con escenas de alta calidad totalmente interactivas, además de cientos de grandes escenas en 3D. Algunas de las características que se incluyen son la aleatorización de dominios, la integración con planificadores de movimiento y herramientas para recopilar datos. Con estas escenas y características, AI2-THOR permite evaluar agentes robóticos que usan señales visuales para resolver tareas de navegación y manipulación. Como son abrir puertas, recoger y colocar objetos o buscar en gabinetes.

La figura 7.3 muestra las imágenes generadas por el simulador y su motor de visualización.

(a) (b)

(c)

Figura 7.3: Ejemplo de imágenes generadas por el simulador AI2-THOR [20]. a) Vista en el robot (textura fotorrealista). b) Mapa de profundidad. c) Vista auxiliar de planta (textura fotorrealista).

Capítulo 8

Pruebas y Resultados

En este capítulo se describe el protocolo de pruebas utilizado y los resultados del sistema completo. El protocolo de pruebas se comenta en la sección 8.1, utilizando un ambiente virtual para recolectar conocimiento del ambiente. Después, en la sección 8.2 se hace un resumen de los resultados de la investigación para la búsqueda de imágenes similares y los resultados de navegación condicionados a objetivos visuales.

8.1. Protocolo de pruebas

Como se describió en la sección 3.4. Los sistemas de aprendizaje por refuerzo requieren de una señal de recompensa generada por el ambiente y de la interacción con el entorno para maximizar una función. Esta función aproxima una estimación de la secuencia de acciones para obtener un recompensa máxima, a partir del estado actual. Debido a esto, la aplicación de aprendizaje reforzado requiere de un ambiente con el cual interactuar, mecanismos de percepción del ambiente y de generar una señal de recompensa. Es una práctica común utilizar ambientes simulados con este propósito, ya que se tienen ventajas respecto al tener sistemas con componentes reales. La mayor utilidad es la de no hacer pruebas destructivas en robots reales, probar el desempeño de los algoritmos antes de transferirlos a un agente real y de ser posible, explotar la paralelización del ambiente simulado. Lo que permite agilizar el entrenamiento de sistemas que tienen el objetivo de ejecutar tareas de manipulación de objetos o de navegación.

En el caso de este trabajo de investigación, se ha utilizado el simulador AI2-THOR [20]. Este simulador es un software desarrollado con el fin de proporcionar un entorno integrado para pruebas de manipulación, navegación visual e integración de percepción visual en sistemas inteligentes.

El simulador es una implementación en el motor de videjuegos Unity [46] con una interfaz de software desarrollada en python. El software se distribuye como una biblioteca que se instala mediante cualquier gestor de paquetes de un entorno de ejecución de lenguaje python. La arquitectura del simulador se muestra en la figura 8.1.

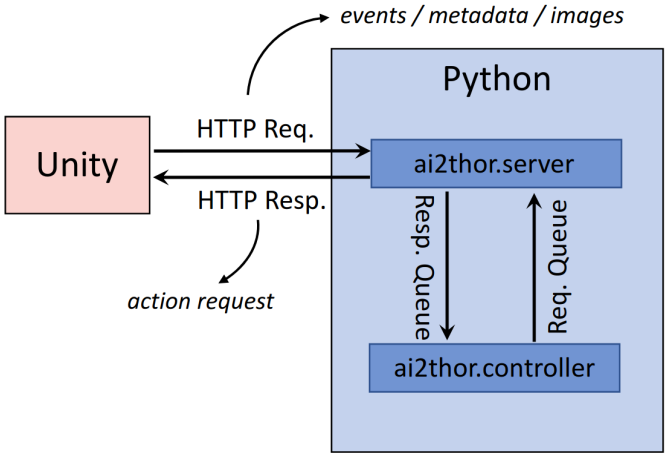
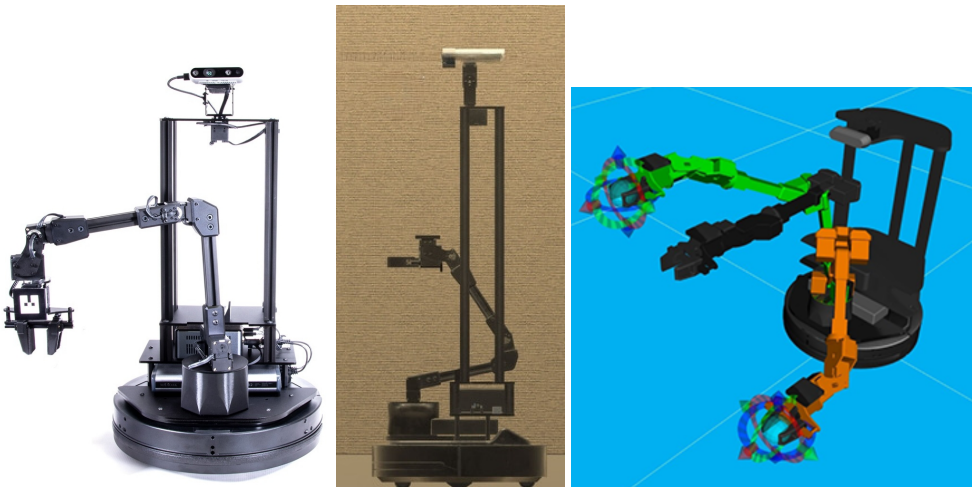


Figura 8.1: Arquitectura del simulador THOR [20] (imagen tomada de [20]).

El simulador incluye 120 escenas como datos de entrenamiento en total, las cuales incluyen información sobre los objetos. Como son sus coordenadas y segmentación semántica, mapas de ocupación además de los modelos virtuales para interacción. Este es el caso de algunos muebles, gabinetes y electrodomésticos. Este simulador incluye información para realizar pruebas de rendimiento como coordenadas donde colocar objetos, regiones de llegada para probar sistemas de navegación, y las funciones de recompensa para poder utilizar sistemas de aprendizaje reforzado con el simulador. Por último, este simulador incluye dos modelos de robots virtuales los cuales son un modelo genérico y una replica en Unity del robot *LocoBot* [28]. Esto tiene el fin de tener la opción de exportar los comportamientos para pruebas reales. Una vista de dicho robot se muestra en la figura 8.2.

El sistema de navegación se evalúa tomando en cuenta la longitud de las trayectorias recorridas, el número de ellas y un punto de destino con un radio definido para obtener una región de tolerancia para el arribo. Existen dos métricas principales para evaluar la navegación de un robot. La primera métrica es el valor de *tasa de éxito* (TE) y se define como:



(a) (b) (c)

Figura 8.2: Robot LocoBot [28] (a)Robot real. (b) Modelo virtual incluido en el simulador AI2-THOR. (c) Modelo virtual genérico.

$$TE = \frac{1}{N}\sum_{i=1}^{N} S_i \tag{8.1}$$

Donde $i \in N$, con N el número de ejecuciones o trayectorias de prueba. La variable S_i es un número binario que indica el éxito del recorrido i, esta variable es determinada por el simulador o se puede calcular usando el mapa de ocupación.

La segunda métrica es el valor de *éxito ponderado por la longitud* (EPL) definida como:

$$EPL = \frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{max\{p_i, l_i\}} \tag{8.2}$$

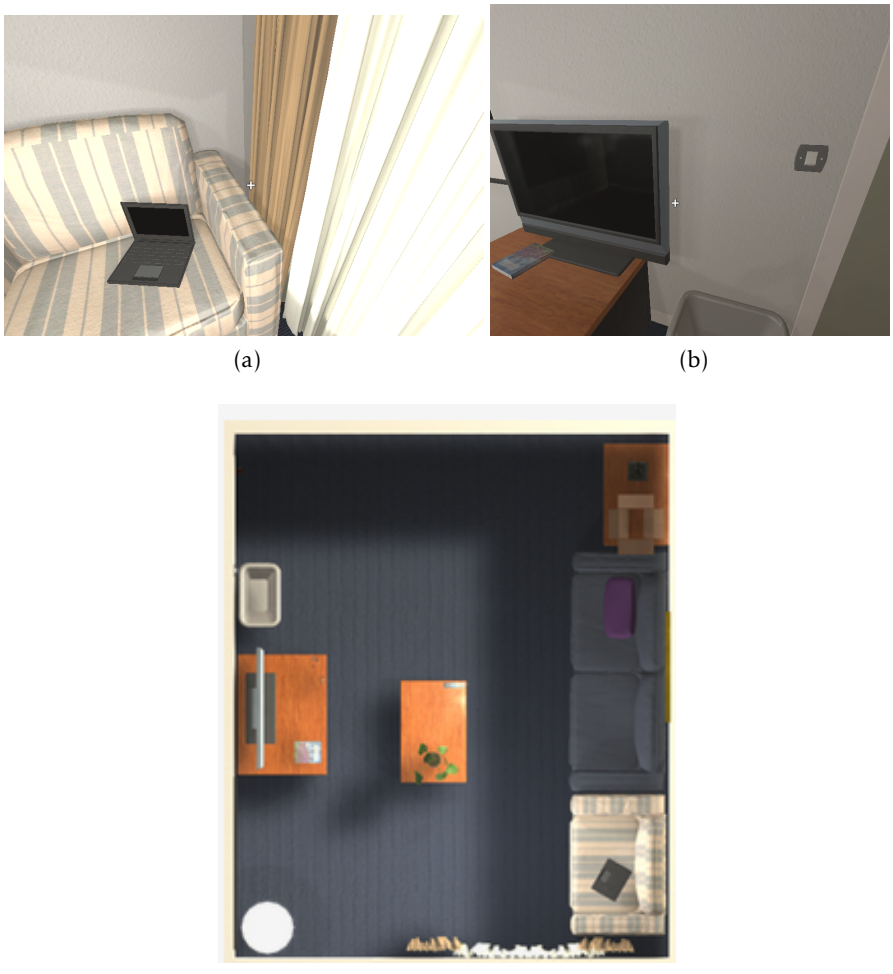
Donde el valor de l_i es la longitud más corta entre el inicio y el objetivo en un episodio. Esta trayectoria es provista en el conjunto de datos como una secuencia de puntos junto con su longitud, o se puede calcular usando un mapa de ocupación. El valor p_i es la longitud de la trayectoria que se evalúa. El ńumero S_i es una variable binaria que indica el éxito del recorrido i, este es un valor '0' o '1' que calcula el simulador.

Finalmente, $EPL \in [0, 1]$ son los valores posibles para la métrica.

Se establece utilizar acciones de salida discretas las cuales son: girar 30 grados; avanzar 0.25 m; un radio de tolerancia entorno al destino de 0.2 m. Adicionalmente el simulador calcula si el robot está orientado hacia el destino.

La señal de recompensa utilizada se fija con los siguientes valores: $r = -0.01$ para una acción no exitosa; y $r = 10$ si se llega al objetivo al menos a 0.2 m de distancia radial además de estar en el campo visual del robot simulado.

El simulador provee 120 escenas como datos. Estas escenas se dividen en 80 escenas de entrenamiento con un total de 12,000 objetivos visuales. Para las pruebas se seleccionan 40 escenas con un total de 2,400 objetivos. Los objetivos incluyen objetos de interés como en escenas son electrodomésticos, muebles, electrónicos, puertas de entrada o salida, y lugares de útiles para colocar o buscar objetos como mesas o anaqueles.



(a) (b)

(c)

Figura 8.3: Ejemplo de escena simulada en el simulador AI2-THOR[20]. (a) Imagen de origen. (b) Imagen de destino. (c) Toma de planta.

8.2. Resultados

En las siguientes dos secciones se enlistan los resultados obtenidos como parte de este trabajo de investigación. Para la sección 8.2.1 son los resultados de búsqueda de imágenes en los dos tipos de memoria construidos. Para la sección 8.2.2 son los resultados del sistema completo, utilizando aprendizaje por refuerzo.

8.2.1. Precisión de búsqueda en imágenes

Para medir la funcionalidad de la memoría por similitud visual se construyó la memoria en las 120 escenas disponibles en el simulador. Usando 256 puntos obtenidos de cada mapa, a partir del mapa de ocupación. Cada punto recolecta imágenes cada 5 grados de rotación, pero solo se usan las imágenes tomadas cada 15 grados para construir la memoria por similitud visual.

Una vez construida la memoria de similitud visual como un grafo, y usando imágenes no entrenadas para la memoria, se hacen pruebas de correspondencia. Los resultados se muestran en la tabla 8.1.

Método	Número de imágenes (5 grados)	Imágenes entrenadas (15 grados)	Imágenes de prueba	Precisión de consulta	Recuperación de consulta
Modelo oculto de Markov	2211840	737280	221160	0.94	0.93
Tabla HSL	2211840	737280	221160	0.98	0.98

Tabla 8.1: Precisión en la búsqueda dentro de la memoria por similitud visual.

El primer renglón en la tabla 8.1 muestra la precisión para detectar la imágen correcta en el grafo, para el caso de utilizar modelos ocultos de Markov (MOM) para describir el contenido visual. El segundo renglón muestra la precisión usando códigos binarios de 256 bits y tablas hash sensibles a la localidad (HSL). Estos dos métodos de construcción del grafo de similitud visual se probaron en un sistema de aprendizaje por refuerzo. Este método se desdcribe en la siguiente sección.

8.2.2. Resultados de navegación visual

Se realizarón pruebas con la arquitectura de la figura 7.2 con tres variantes: la primera variación es utilizar ResNet-18 como extractor de características; la segunda variante es

usar ResNet-50; la tercera es usar la firma binaria de 256 bits usada para crear el método de hash sensible a la localidad. A estas variaciones se les probó con el método de construcción de memoria usando modelos ocultos de Markov (MOM) y usando hash sensible a la localidad (HSL).

Se utiliza la *optimización de políticas próximas* (OPP) como método de entrenamiento. Este método es utilizado debido a su rápida convergencia respecto otros métodos y funciona para acciones de salida discretas o continuas. Además, este método permite utilizar varios simuladores de manera concurrente haciendo instancias de la arquitectura para recolectar más pasos de exploración del ambiente.

Óptimización de políticas próximas se define como sigue [47]:

$$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}_{s,a\sim\pi_{\theta_k}}[L(s,a,\theta_k,\theta)] \tag{8.3}$$

$$L(s,a,\theta_k,\theta) = \min\left\{\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)}A^{\pi_{\theta_k}}(s,a),\ g\left[\epsilon, A^{\pi_{\theta_k}}(s,a)\right]\right\}$$

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A, & A \geq 0 \\ (1-\epsilon)A, & A < 0 \end{cases}$$

Y se usa la aproximación definida de la siguiente manera:

$$A^{\pi_{\theta_k}}(s_t, a_t) \approx \hat{A}(s_t;\theta_v) \tag{8.4}$$

$$\hat{A}(s_t;\theta_v) = \sum_{i=0}^{T-1} \gamma^i r_{t+1} + \gamma^i V(s_{t+i};\theta_v) - V(s_t;\theta_v)$$

Se estableció una recompensa positiva en r=10 y negativa en r=-0.01. El entrenamiento se limita a $6x10^6$ imágenes. La tasa de aprendizaje varia desde el valor 0.01 hasta 0.0001 con decaimiento lineal. El método de óptimización es *AdaGrad*[48] para OPP. Se usan 8 procesos para simular las interacciones en paralelo.

Las acciones utilizadas son discretas: avanzar 0.25 m; girar 30 grados a la izquierda o derecha; retroceder 0.25 m y detenerse.

Se agrega ruido gaussiano en las acciones de desplazamiento con parámetros $\mu = 0$ y σ =0.02 m y rotación con parámetros $\mu_r = 0$ y σ_r=0.5 grados.

La tabla 8.2 muestra el consumo de memoria de GPU por cada método basado en ResNet utilizando hasta 8 procesos simultáneos del simulador, para el caso de ResNet-18. En el caso de ResNet-50, hay que disminuir este valor para poder alojar el entrenamiento.

	ResNet-18	ResNet-50
Número de parámetros	11.4x10^6	27.5x10^6
Memoria GPU 1 agente	83.5 MB	209.8 MB
Memoria GPU 1 agente 2048 imágenes	1.23 GB	3.75GB
Memoria GPU 8 agentes 2048 imágenes $^{por}/_{agente}$	10.84 GB	*

Tabla 8.2: Consumo de recursos en GPU (*sin memoria suficiente).

EL GPU utilizado es el modelo RTX 2080 Ti con 11.2 GB de memoria para alojar datos. El equipo utilizado es una estación de trabajo con 64 GB de memoria RAM y procesador central Ryzen 5950x de 32 hilos de ejecución.

La tabla 8.3 muestra el consumo de recursos de memoria. Para describir una imágen y agregarla a una tabla HSL se requiere calcular la firma binaria, esto se muestra en el primer reglón.

Para aplicar el método publicado en [40], es necesario describir las imágenes en el espacio de color CIE Lab y hacer los vectores de entrada para 3 clasificadores en conjunto. El segundo renglón muestra el consumo de memoria para procesar 10^6 imágenes, debido a que el método es muy compacto en memoria es posible crear tablas HSL de gran tamaño.

La memoria visual creada por escenas, captura imágenes cada 15 grados de rotación y para 256 puntos muestreados del mapa de ocupación de dicha escena. Esto es $256 \times 24 = 6144$ imágenes por escena. Se crean 5 aristas por cada imágen según su probabilidad de similitud dando un total de $6114x5 = 30720$ aristas en cada grafo.

En el caso del sistema de aprendizaje reforzado, se utilizan tres indicadores de la convergencia del sistema: distancia promedio de todos los objetivos entrenados; tasa de éxito o valor TE, que son los conteos en los cuales el agente simulado llega a su objetivo; la recompensa acumulada durante el entrenamiento.

La figura 8.4 muestra la distancia promedio a los objetivos entrenados para la arqui-

	Memoria con HSL	Memoria con MOM
Firma para 2048 imágenes	64 KB $k^2 = 256$ bits	9.2GB 3,6,9 estados por imagen
Tabla HSL 10^6 imágenes	42.7 GB	*
Tabla HSL $200x10^3$ imágenes	8.3 GB	*

Tabla 8.3: Consumo de memoria del sistema por método (*sin memoria suficiente).

tectura basada en ResNet-18 y memoria visual construida con el método HSL. El comportamiento esperado, es que la distancia promedio tienda al promedio de todas las trayectorias más cortas.

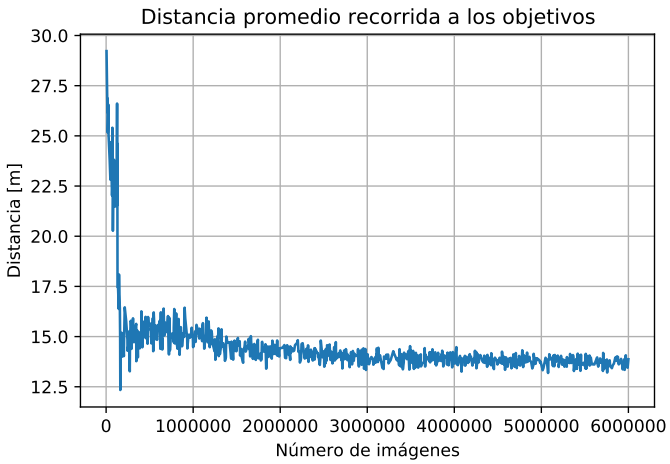


Figura 8.4: Distancia promedio a los objetivos entrenados. Arquitectura basada en ResNet-18 y memoria construida con HSL.

La figura 8.5 muestra la distancia promedio a los objetivos entrenados para la arquitectura basada en ResNet-50 y memoria visual construida con el método HSL.

La figura 8.6 muestra la tasa de éxito promedio a los objetivos entrenados para la arquitectura basada en ResNet-18 y memoria visual construida con el método HSL. El comportamiento esperado es que la tasa de éxito, tienda al promedio de todas las trayectorias que llegan al objetivo visual.

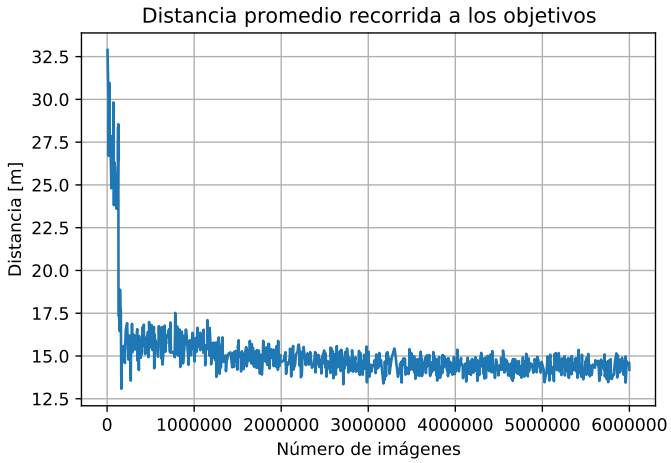


Figura 8.5: Distancia promedio a los objetivos entrenados. Arquitectura basada en ResNet-50 y memoria construida con HSL.

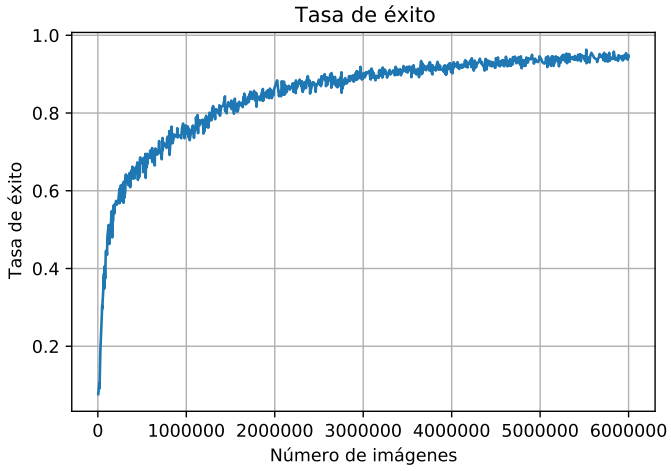


Figura 8.6: Valor de tasa de éxito. Arquitectura basada en ResNet-18 y memoria construida con HSL.

La figura 8.7 muestra la tasa de éxito promedio a los objetivos entrenados, para la arquitectura basada en ResNet-50 y memoria visual construida con el método HSL.

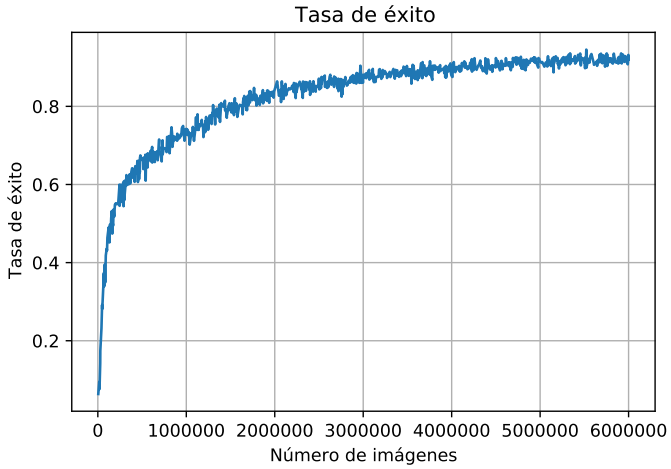


Figura 8.7: Valor de tasa de éxito. Arquitectura basada en ResNet-50 y memoria construida con HSL.

El último indicador es la recompensa acumulada, que es la suma de recompensas por episodio. En este trabajo un episodio corresponde a una trayectoria hacia el objetivo. Conforme el agente simulado aprende a ejecutar acciones, este valor tiende al promedio de recompensas acumuladas posibles.

La figura 8.8 muestra la recompensa acumulada para la arquitectura basada en ResNet-18. Conforme el sistema aumenta la cantidad de acciones ejecutadas y recolecta imágenes nuevas, el sistema mejora el aprendizaje de la política de acción. Este comportamiento se observa como una función siempre creciente por lo que se establece una condición de paro en el entrenamiento con la cantidad total de imágenes procesadas, por cada instancia de un agente simulado.

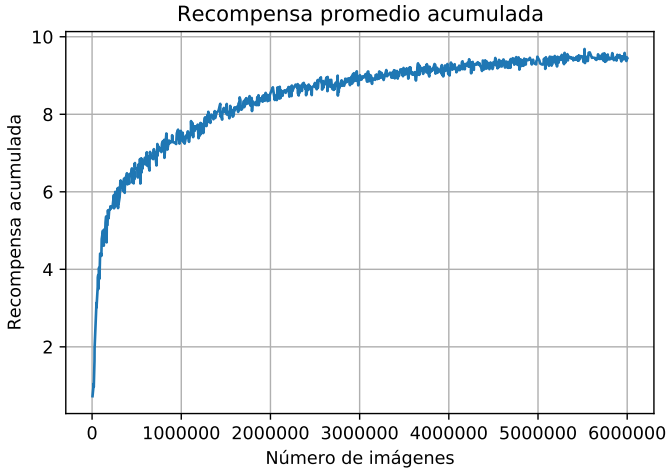


Figura 8.8: Recompensa acumulada. Arquitectura basada en ResNet-18 y memoria construida con HSL.

La figura 8.9 muestra la recompensa acumulada promedio para la arquitectura entrenada con ResNet-50. Se observa con respecto a la figura 8.8, que solo tiene un tendencia hacia un valor menor de 8, cerca de los 2 millones de imágenes. En cambio la arquitectura basada en ResNet-18 rebasa ese valor cerca de los 2 millones de imágenes.

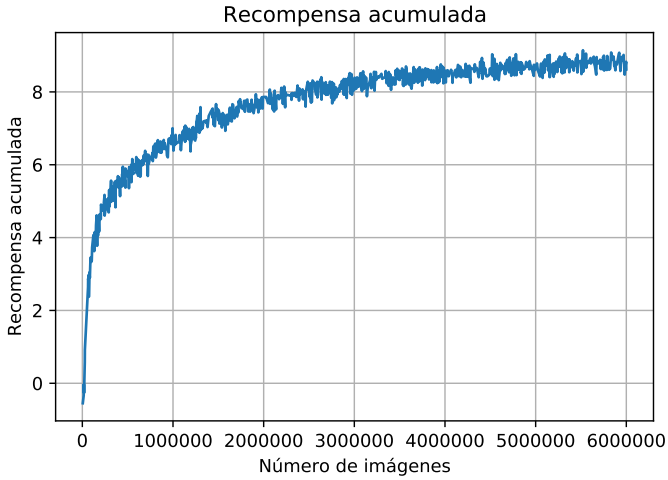


Figura 8.9: Recompensa acumulada. Arquitectura basada en ResNet-18 y memoria construida con HSL.

Las dos principales métricas para medir el desempeño de un sistema de navegación visual es el valor TE y el valor EPL. TE es el promedio de trayectorias en las que se logra

llegar al objetivo, sin embargo este valor no determina que tan larga es la longitud ejecutada. Por esto, el valor EPL incopora la longitud de una trayectoria óptima calculada internamente por el simulador y un planeador basado en el algoritmo Dijkstra [49].

La tabla 8.4 muestra la evaluación del conjunto de entrenamiento, para todas las variaciones probadas. Para el caso de la memoria construida con tablas HSL se indica con las letras m_1. Para el caso del uso de la memoria construida con modelos ocultos de Markov (MOM) se escribe m_2 para indicarlo.

Modelo	TE	EPL
ResNet-18, m_1	0.923	0.834
ResNet-18, m_2	0.852	0.764
ResNet-50, m_1	0.842	0.723
ResNet-50, m_2	0.788	0.642
Firma binaria, m_1	0.653	0.568
Firma binaria, m_2	0.564	0.533

Tabla 8.4: Métricas obtenidas para el conjunto de entrenamiento. Memoria construida con tablas HSL se indica con las letras m_1. Memoria construida con MOM se indica con las letras m_2.

La tabla 8.5 muestra la evaluación del conjunto de pruebas para todas las variaciones. Para el caso del uso de la memoria construida con modelos ocultos de Markov (MOM) se escribe m_1 para indicarlo. Para el caso de la memoria construida con tablas HSL se indica con las letras m_2.

Modelo	TE	EPL
ResNet-18, m_1	0.624	0.203
ResNet-18, m_2	0.571	0.193
ResNet-50, m_1	0.562	0.182
ResNet-50, m_2	0.564	0.156
Firma binaria, m_1	0.532	0.142
Firma binaria, m_2	0.524	0.108

Tabla 8.5: Métricas obtenidas para el conjunto de pruebas. Memoria construida con tablas HSL se indica con las letras m_1. Memoria construida con MOM se indica con las letras m_2.

En la tabla 8.5 correspondiente a los resultados del conjunto de pruebas. Se observa en el primer renglón que el sistema basado en ResNet-18 con la memoria construida con tablas HSL, tiene el mejor desempeño obteniendo un valor TE de 0.624 y un EPL de 0.203.

La segunda variante es utilizando la construcción de memoria con modelos ocultos de Markov. Este resultado muestra que el uso de una memoria auxiliar mejora el aprendizaje

de políticas de navegación, utilizando ResNet-18 como extractor de características visuales. Para el caso de ResNet-50, se observa un comportamiento similar para la memoria de tipo m_1, aunque el desempeño es menor en las métricas.

Tambien, se probó el utilizar la firma binaria directamente como entrada al sistema de navegación. Pero se obtienen resultados inferiores al de la arquitectura propuesta como se muestra en los dos últimos renglones de la tabla 8.2 y la tabla 8.3.

Capítulo 9

Conclusiones y Trabajo Futuro

En este trabajo de investigación se desarrolló un sistema de navegación visual utilizando el simulador AI2-THOR para realizar pruebas en un robot simulado. El objetivo fue el de incoporar una memoria del ambiente a un arquitectura basada en aprendizaje reforzado. Usando redes neuronales convolucionales como extractores de características visuales fue posible describir información para representar el estado del ambiente. Se probaron seis tipos distintos de entradas con características visuales, para una arquitectura definida que implementó el aprendizaje por refuerzo.

Se comprobaron las hipótesis planteadas al inicio de la sección 1.3 las cuales son:

- Para la hipótesis: *"La navegación en un robot móvil puede llevarse a cabo solamente utilizando técnicas de aprendizaje profundo para imágenes, aprendizaje por refuerzo e información topológica por similitud visual. Dando lugar a diseñar un sistema con memoria auxiliar visual del ambiente y navegación"*, se comprobó que dicha propuesta resulta en un desempeño mejor junto con la arquitectura basada en ResNet-18. Esto es una ventaja, ya que la eficiencia del uso de datos es un problema común en sistemas basados en redes neuronales convolucionales. De los resultados obtenidos es posible concluir que el sistema que utilizó una memoria del ambiente y ResNet-18 como extractor de características visuales obtuvó el mejor rendimiento en las métricas TE y EPL.

- Para la hipótesis: *"Se puede construir una memoria visual auxiliar usando dos métodos de clasificación probabilística para entrenar un sistema basado en aprendizaje por refuerzo con mejor desempeño. Los métodos son modelos ocultos de Markov (MOM) y hash sensible a la localidad (HSL), ya que ambos métodos establecen un modelo probabilístico del contenido en las imágenes agregando robustez al sistema"*, se comprueba que

es posible realizar una memoria por similitud visual, que representa información topológica del ambiente e insertar dicha memoria como componente de un sistema de aprendizaje por refuerzo con redes neuronales. Es comparación para el mismo tipo de construcción de la memoría, y el mismo método de óptimización, se obtiene un mejor desempeño para el caso de tablas HSL. Siendo mejor en el caso de la arquitectura que usa ResNet-18.

Para el caso de ResNet-50 y el mismo tipo de memoria, en comparación con las pruebas de ResNet-18. Se observa un rendimiento menor según se reportó en las tabla 8.2 y la tabla 8.3. Esto lleva a la conclusión de que igualmente la dimensión de las características visuales de entrada al sistema de aprendizaje por refuerzo tienen un impacto en la velocidad de convergencia y los valores TE y EPL obtenidos. Esto resultados pueden asociarse a las variaciones en las trayectorias generadas con estas arquitectura como se observa en la figura 8.4 y la figura 8.5, en donde la segunda figura tiene valores más amplios en la distancia promedio a sus objetivos. Este efecto tambien es apreciable en las figuras 8.6 y 8.7, esta última con un crecimiento más lento en comparación con la arquitectura ResNet-18.

También es posible justificar el uso de información topológica en una forma no paramétrica, aliviando el incremento de parámetros en el sistema mediante un módulo externo de software y utilizando el sistema de mapeo interno del simulador o utilizando alguno otro si fuera posible su integración.

Finalmente, los trabajos académicos siguientes fueron desarrollados y publicados como consecuencia del enfoque de usar una memoria no paramétrica para integrar información del ambiente.

- *Comparison of Two Objects Classification Techniques using Hidden Markov Models and Convolutional Neural Networks.* Sarmiento Carlos, Savage Jesus. Informatics and Automation (SPIIRAS Proceedings 2020). Vol 19. No 6. Diciembre 2020.
  https://doi.org/10.15622/ia.2020.19.6.4
  (Indexado en Scopus).

- *Feature detection using Hidden Markov Models for 3D-visual recognition.* Carlos Sarmiento, J. Savage, et al. IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC2019).
  https://doi.org/10.1109/ICARSC.2019.8733651
  (Indexado en Scopus).

El sistema final que se ha documentado en este trabajo presenta algunas ventajas sobre sistemas basados en localización y mapeo simultáneo (LMS), estas son:

- Un sistema entrada-salida para navegación visual.
  En comparación con sistemas localización y mapeo simultáneo (LMS) o rastreo y mapeo en paralelo (RMP) se sustituye la deteccion de puntos y su rastreo, junto con todos los subsistemas por técnicas de aprendizaje profundo en un solo sistema.

- Sistema basado en exploración y un objetivo visual.
  En sustitución de un mapa, se utiliza una red profunda como mecanismo de retención de experiencia y a su vez, esta ejecuta una política de selección de acciones. Esto evita la construcción de subsistemas de planeación dependientes de LMS o RMP.

- El mecanismo de entrenamiento permite la fusión de fuentes de información como es el estado actual y el destino. Por lo que el sistema puede ser expandido a nuevos tipos de sensores o entradas. Para el caso de LMS o RMP es necesario replantear el problema de optimización.

9.1. Trabajo futuro

Las pruebas realizadas fueron hechas en un modelo de robot no disponible en el laboratorio conocido como LocoBot, pero uno de los objetivos de los sistemas virtuales que incluyen cinemática y dinámica, es el de transferir las políticas de navegación a un agente real para pruebas. Como parte del trabajo futuro, se plantea la incoporación de modelos de robots disponibles en el Laboratorio de Biorrobótica. Dos de estos modelos son del robot Justina y el robot HSR de la empresa Toyota. Estos dos modelos de robots son desarrollados para el simulador *Gazebo*, pero se sabe que Gazebo es una herramienta commputacionalmente demandante. En simulaciones típicas, resulta poco común poder obtener un factor de tiempo igual a 1, lo que significa una simulación en tiempo real. Esto se debe a que el simulador debe resolver sistemas de ecuaciones diferenciales varias veces por segundo y actualizar el estado de todos los modelos. En comparación el motor de física de Unity está altamente optimizado, ya que incorpora tecnología patentada no disponible en el motor *Bullet*, con el que se desarrollan algunos componentes de simulación de Gazebo.

Respecto al sistema simulación, actualmente no se cuenta con un ambiente fotorrealista del laboratorio, por lo que esto puede ser una mejora para probar el sistema aquí desarrollado en un sistema virtual lo más realista posible. Una manera de realizar esto es utilizar bibliotecas de fotogrametría como son las incorporadas en el software*Blender*. Que es un software de modelado en 3D, pero sus bibliotecas pueden conectarse y operarse mediante código en lenguaje python. Lo que resulta en la posibilidad de realizar una

reconstrucción 3D de un ambiente fotografiado. Para obtener un modelo de polígonos y ser incorporado en el motor de simulación de AI2-THOR.

En el caso de querer aplicar este sistema de navegación en un ambiente competitivo, como puede ser un entorno en torneos de robótica, ya no sería de manera simple utilizar el entrenamiento virtual. Para esto, sería necesario recurrir a una capa de extracción de información semántica como es un detector de objetos o personas y reemplazar las entradas por las coordenadas de los objetos detectados o las personas, estas coordenadas pueden estar concantendas o ser usadas de manera independiente. Esto sería una mejora potencial, la cual aún requiere de recolectar imágenes en el ambiente. También es posible reemplazar las entradas por una nube de puntos segmentada a un rango fijo de distancia, o el usar detectores en nubes de puntos con redes convolucionales, ya que estos incoporan permutaciones de las entradas aportando robustez en un escenario real.

Dado un entrenamiento en un sistema virtual con cientos de distintos ambientes, es una opción de investigación el construir una digitalización de un ambiente real como una arena de pruebas o el laboratorio actual. Para transferir y reentrenar el sistema de navegación visual a este ambiente específico. Con esto se reduciría la brecha entre un sistema entrenado virtualmente y uno real.

Otra mejoras para el sistema de aprendizaje reforzado pueden ser utilizar GPUs distribuidos en dos equipos de trabajo, ya que los resultados expuestos en este trabajo solo consideran el uso de GPUs accesibles en la estación de trabajo. Pruebas con distintos métodos de gradiente descendiente u métodos distribuidos puede resultar en una convergencia aún más ágil.

Finalmente es posible portar este sistema a hardware de propósito específico como son los procesadores Jetson AGX®, Jetson nano® y de la familia de productos de NVIDIA® para la industria. En el caso de computadores personales existen soluciones de la empresa Intel®, como OpenVINO® que son coprocesadores tensoriales con conexión USB 3.0.

Bibliografía

[1] J. Engelberger, *Robotics in Service*. MIT Press, 1989.

[2] K. H. Park, H. E. Lee, Y. Kim, and Z. Z. Bien, "A steward robot for human-friendly human-machine interaction in a smart house environment," *IEEE Transactions on Automation Science and Engineering*, vol. 5, pp. 21–25, Jan 2008.

[3] D. Montemello, *The Cambridge Handbook of Visuospatial Thinking*, ch. Navigation. Cambridge University Press, 2005.

[4] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, (San Francisco, CA, USA), pp. 674–679, Morgan Kaufmann Publishers Inc., 1981.

[5] S. D. Jones, C. Andresen, and J. L. Crowley, "Appearance based process for visual navigation," in *Intelligent Robots and Systems, 1997. IROS '97., Proceedings of the 1997 IEEE/RSJ International Conference on*, vol. 2, pp. 551–557 vol.2, Sep 1997.

[6] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. New York, NY, USA: Cambridge University Press, 2 ed., 2003.

[7] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[8] A. Concha and J. Civera, "An evaluation of robust cost functions for rgb direct mapping," in *2015 European Conference on Mobile Robots (ECMR)*, pp. 1–8, 2015.

[9] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 225–234, 2007.

[10] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[11] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 834–849, Springer International Publishing, 2014.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[13] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[14] H. F. Ólafsdóttir, D. Bush, and C. Barry, "The role of hippocampal replay in memory and planning," *Current Biology*, vol. 28, no. 1, pp. R37 – R50, 2018.

[15] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," *CoRR*, vol. abs/1706.01905, 2017.

[16] L. Wang, L. Zhao, G. Huo, R. Li, Z. Hou, P. Luo, Z. Sun, k. Wang, and C. Yang, "Visual semantic navigation based on deep learning for indoor mobile robots," *Complexity*, vol. 2018, pp. 1–12, 04 2018.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *ArXiv e-prints*, Dec. 2015.

[18] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[19] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, M. Bernstein, and L. Fei-Fei, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," 2016.

[20] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, "AI2-THOR: an interactive 3d environment for visual AI," *CoRR*, vol. abs/1712.05474, 2017.

[21] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell, "Learning to navigate in complex environments," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[22] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[23] X.-H. Le, H. V. Ho, G. Lee, and S. Jung, "Application of long short-term memory (lstm) neural network for flood forecasting," *Water*, vol. 11, no. 7, 2019.

[24] X. Zhou, Y. Gao, and L. Guan, "Towards Goal-Directed Navigation Through Combining Learning Based Global and Local Planners.," *Sensors (Basel, Switzerland)*, vol. 19, jan 2019.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[26] J. Savage, D. A. Rosenblueth, M. Matamoros, M. Negrete, L. Contreras, J. Cruz, R. Martell, H. Estrada, and H. Okada, "Semantic reasoning in service robots using expert systems," *Robot. Auton. Syst.*, vol. 114, p. 77–92, apr 2019.

[27] A. Merzlyakov and S. Macenski, "A comparison of modern general-purpose visual slam approaches," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[28] A. Murali, T. Chen, K. V. Alwala, D. Gandhi, L. Pinto, S. Gupta, and A. Gupta, "Pyrobot: An open-source robotics framework for research and benchmarking," *arXiv preprint arXiv:1906.08236*, 2019.

[29] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, (Cambridge, MA, USA), pp. 1057–1063, MIT Press, 1999.

[30] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1st ed., 1996.

[31] L. Rabiner and B. Juang, "An introduction to hidden markov models," *IEEE ASSP Magazine*, vol. 3, pp. 4–16, Jan 1986.

[32] T. Mailund, *The Joys of Hashing: Hash Table Programming with C*. Apress, 2019.

[33] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, "Practical and optimal lsh for angular distance," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.

[34] A. Broder, "On the resemblance and containment of documents," in *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pp. 21–29, 1997.

[35] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*. USA: Cambridge University Press, 2nd ed., 2014.

[36] I. M. Hameed, S. H. Abdulhussain, and B. M. Mahmmod, "Content-based image retrieval: A review of recent trends," *Cogent Engineering*, vol. 8, no. 1, p. 1927469, 2021.

[37] I. M. Hameed, S. H. Abdulhussain, and B. M. Mahmmod, "Content-based image retrieval: A review of recent trends," *Cogent Engineering*, vol. 8, no. 1, p. 1927469, 2021.

[38] "Yolov4: Optimal speed and accuracy of object detection." https://github.com/AlexeyAB/darknet. Accessed: 2021-12-20.

[39] X. Li, J. Yang, and J. Ma, "Recent developments of content-based image retrieval (cbir)," *Neurocomputing*, vol. 452, pp. 675–689, 2021.

[40] C. Sarmiento and J. Savage, "Comparison of two objects classification techniques using hidden markov models and convolutional neural networks," *Informatics and Automation*, vol. 19, pp. 1222–1254, Dec. 2020.

[41] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[42] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1928–1937, PMLR, 20–22 Jun 2016.

[43] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[44] C. Li, F. Xia, R. Martín-Martín, M. Lingelbach, S. Srivastava, B. Shen, K. E. Vainio, C. Gokmen, G. Dharan, T. Jain, A. Kurenkov, K. Liu, H. Gweon, J. Wu, L. Fei-Fei, and S. Savarese, "igibson 2.0: Object-centric simulation for robot learning of everyday household tasks," in *5th Annual Conference on Robot Learning*, 2021.

[45] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A Platform for Embodied AI Research," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[46] J. K. Haas, "A history of the unity game engine," 2014.

[47] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[48] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, p. 2121–2159, jul 2011.

[49] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.