uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Informática

2020-2021

Trabajo Fin de Grado

"Aprendizaje por refuerzo para la navegación autónoma de drones para el mapeo tridimensional de entornos"

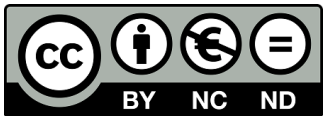
Pablo Cañadas Miquel

Tutores

Antonio Berlanga de Jesús

Miguel Ángel Patricio Guisado

Colmenarejo, 21 de septiembre de 2021



RESUMEN

Gracias a los últimos avances en tecnologías como LIDAR y UAVs, han aparecido soluciones de mapeo de terrenos tridimensionales de alta precisión para entornos. Aunque actualmente existen soluciones que permiten obtener mapas con alto nivel de detalle, suelen ser en vehículos pesados y con costes elevados y no suelen permitir el mapeo de todo tipo de entornos como por ejemplo interiores cerrados.

En este trabajo se investigan las tecnologías LIDAR, UAVs y estado del arte en navegación autónoma de UAVs, y bajo futuras expectativas de mejora en las tecnologías, se estudia una solución que, aprovechando la precisión de LIDAR y la potencia del aprendizaje automático, permitiría mapear entornos de forma autónoma con un buen nivel de detalle en drones de reducido tamaño.

En primer lugar se diseña e implementa una herramienta eficiente para el mapeo tridimensional con LIDAR que construye un mapa tridimensional en tiempo real. Esta herramienta usa estructuras y algoritmos eficientes para permitir su ejecución en microprocesadores que los drones puedan portar y además permitiendo minimizar el tiempo de entrenamiento del agente del sistema de navegación autónoma.

Para el sistema de navegación autónoma se propone el uso de aprendizaje por refuerzo para resolver el reto de combinar una exploración global del entorno con una exploración local de alto nivel de detalle. Se apuesta por aprendizaje por refuerzo debido a que ha demostrado previamente buen rendimiento en problemas similares.

Se implementa una solución final en simulador y demuestra su funcionamiento sirviendo como una prueba del concepto. Para esto se investigan, combinan y prueban ideas y software que pueden ser reutilizados para problemas similares.

El agente autónomo creado no muestra resultados realmente prácticos, pero el trabajo realizado presenta bases para trabajos futuros y los resultados ofrecen información y sugerencias sobre las dificultades prácticas de resolver el problema. Además, este trabajo ofrece diseños de herramientas eficientes para el mapeo de entornos con drones.

Palabras clave: UAVs, Mapeo 3D, LIDAR, Aprendizaje por refuerzo.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

1. INTRODUCCIÓN

Gracias a los avances de los últimos años en los vehículos aéreos no tripulados (VANTs ó UAVs), las técnicas de mapeo de terrenos han ido evolucionando haciendo uso de estos vehículos y en la actualidad existen casos de uso de UAVs que, utilizando sensores LIDAR, cámaras o ambas, son capaces de generar mapas topográficos de los terrenos que sobrevuelan. Tal es el caso de [1].

Los métodos de navegación para los UAVs, suelen ser manuales o, en algunos casos, completamente automatizados con algoritmos simples que permiten simplemente desplazar el vehículo de un punto a otro. El problema de la automatización para la exploración de terrenos complejos ha visto cierto progreso gracias al desarrollo de técnicas como el aprendizaje por refuerzo que han permitido crear algoritmos de navegación capaces de realizar tareas de búsqueda o reconocimiento.

El uso de LIDAR en UAVs ha aumentado considerablemente gracias a avances en esta tecnología que han reducido su coste, peso y tamaño haciéndolo viable para su uso en vehículos aéreos más pequeños y económicos como los drones. Aunque por el momento los sensores LIDAR más ligeros y pequeños poseen características limitadas para un mapeo de calidad, siguiendo el continuado progreso de la tecnología es de esperar que en un futuro próximo se puedan ver mejoras sustanciales de cara a su uso para mapeo de superficies.

Otro de los problemas a los que se enfrentan los UAVs de pequeño tamaño, es la ausencia de sensores ligeros de posicionamiento de alta precisión, como GPS o IMUs. Esta es históricamente una de las mayores limitaciones a la hora de recabar datos posicionales con UAVs, pero en los últimos años, gracias a los continuos avances en IMUs y ciertas alternativas de GNSS, el uso de UAVs se ha vuelto más fiable y al igual que con la tecnología LIDAR, es de esperar que eventualmente alcancen altos niveles de precisión.

Actualmente el mapeo de terrenos con UAVs se usa principalmente para la generación de mapas de vista aérea usando normalmente fotogrametría con la posibilidad de añadir el uso de LIDAR para mayor precisión. Aunque los mapas generados usando estas técnicas permiten la visualización de entornos desde varias perspectivas, generalmente ofrecen un nivel de detalle tridimensional muy limitado, especialmente desde ángulos horizontales, siendo incapaces de generar mapas tridimensionales de entornos cerrados como, por ejemplo, el interior de un edificio.

El objetivo de este trabajo es diseñar en un simulador un sistema de mapeo tridimensional de entornos e intentar otorgarle un pilotaje completamente autónomo y con alto nivel de detalle y cierta completitud. Para esto se hace uso de drones con hipotéticos sensores LIDAR de alta precisión capaces de cubrir una cúpula entera

de 360^{o} y con alcance de decenas de metros. Aunque por el momento no es viable construir semejante dispositivo, este trabajo se realiza bajo la esperanza de que los futuros avances permitan algo similar y con ello, más que a efectos prácticos, el objetivo de este trabajo es demostrar el concepto y ofrecer una base de trabajo. Dadas las complicaciones de obtener un sistema de navegación completamente autónomo, se hará uso de aprendizaje por refuerzo.

2. OBJETIVOS

Este trabajo busca probar el potencial del uso de los sensores LIDAR para mapear tridimensionalmente entornos tanto abiertos como cerrados de forma autónoma haciendo uso de drones. Aunque previamente se han obtenido soluciones capaces de mapear terrenos de forma no muy detallada y con vista vertical aérea, en este trabajo se busca obtener mapas más detallados y de entornos más variados (incluyendo interiores cerrados), con la capacidad de visualizarlos desde todos los ángulos sin perder detalle.

A causa de las limitaciones de la tecnología, actualmente sólo es posible obtener resultados de alta calidad en simulador, pero gracias a que existen expectativas positivas para el desarrollo de las tecnologías de LIDAR, UAVs y sensores de posicionamiento (GNSS e IMUs), se espera que en el futuro sea posible implementar algo similar en el mundo real.

Con ello, este trabajo pretende ofrecer una base de trabajo para crear un sistema de mapeo tridimensional autónomo para drones ligeros, mostrando posibles maneras de enfrentar los retos que esto conlleva, desde el diseño de herramientas eficientes hasta la creación de un sistema de navegación autónomo.

Para obtener esta solución, se busca diseñar en simulador un sistema que genere y transforme datos de sensores LIDAR recogidos de un dron con un piloto humano o un sistema de pilotaje autónomo. El sistema de pilotaje autónomo será diseñado aprovechando las ventajas del uso del aprendizaje automático, más específicamente del aprendizaje por refuerzo.

El producto final que se busca obtener del sistema diseñado son modelos tridimensionales completos del entorno simulado. Para esto el dron deberá desplazarse bajo unos límites establecidos y recabar información de todas las superficies del entorno.

La meta de este trabajo se puede descomponer en dos apartados:

1. Procesado, almacenamiento y visualización de datos. LIDAR normalmente genera datos en forma de puntos tridimensionales globales ó relativos a la posición y rotación del sensor. El principal problema para tratar estos datos es que el sensor puede generar cantidades muy grandes de datos, donde gran parte de ellos son redundantes y perjudican el rendimiento. Por ello, en este trabajo se diseña una herramienta de mapeo a partir de un algoritmo capaz de decidir si insertar u omitir datos en grandes cantidades.

2. Navegación autónoma del dron. Para obtener un mapeo completo de cualquier tipo de entorno, es necesario crear un algoritmo de navegación capaz de ex-

plorar por sí mismo y adaptarse a entornos desconocidos. Para resolver este problema se usa aprendizaje por refuerzo que ya ha demostrado previamente sus capacidades en otros retos de exploración con drones. De todas las soluciones generadas, se selecciona la que es capaz de generar los mapas más completos y detallados.

3. ESTADO DEL ARTE

3.1. Mapeo de terrenos

La ciencia de extraer información sobre objetos físicos a partir de imágenes fotográficas e imágenes basadas en otros fenómenos (como radar) es generalmente conocida como fotogrametría. Más concretamente, existe la estereofotogrametría que es la rama de la fotogrametría para la reconstrucción de mapas tridimensionales usando imágenes fotográficas [2].

Aunque la estereofotogrametría usando imágenes fotográficas es precisa para extraer medidas horizontales (ejes 'x' e 'y'), estas técnicas por sí solas tienen dificultades para extraer medidas verticales precisas (eje 'z'). LIDAR ha encontrado su uso en esta área para añadir mediciones de altura (topográficas) y mejorar así la precisión vertical de las recreaciones tridimensionales.

Actualmente existe tecnología capaz de reconstruir todo tipo de mapas tridimensionales con alto nivel de detalle usando drones como es el caso del modelo "Elios 2" [3] capaz de generar modelos con hasta 1mm de precisión, pero son drones pesados, voluminosos y costosos que requieren navegación manual.

A nivel comercial, existe un mercado en el que compañías ofrecen software de mapeo para drones, drones con su software específico o incluso soluciones completas por equipos profesionales [4][5].

SLAM

Dentro del mapeo de terrenos con vehículos autónomos destaca un tipo de problema conocido como "Simultaneous Localization And Mapping" (SLAM) que consiste en la construcción de un mapa tridimensional a partir de las observaciones de un vehículo y con ello simultáneamente poder localizarse en este mapa. Estas técnicas destacan por permitir mapear terrenos de forma autónoma [6].

La finalidad original de estos algoritmos era la navegación de robots, pero también es posible generar mapas tridimensionales de cierta calidad.

Usando SLAM destaca el algoritmo diseñado en [7]. Este algoritmo utiliza "monocular SLAM", técnica en la que se usa exclusivamente una cámara RGB para navegación y generación del mapa, y fue diseñado para explorar de forma autónoma entornos cerrados con drones pequeños, generando así mapas tridimensionales de ellos. Este algoritmo destaca por generar mapas con buen nivel de detalle bajo unos reducidos requisitos de computación, pero no garantiza mapas completos, no rinde bien en entornos más grandes y, al usar exclusivamente una cámara rgb, es propenso

a errores de precisión.

3.2. Sensores LIDAR

En este trabajo se usa en simulador la tecnología LIDAR (ó LiDAR) para el mapeo de los entornos, generando nubes de puntos tridimensionales de alto detalle y precisión.

Para comprender el estado del arte en LIDAR y sus expectativas de crecimiento, es necesario comprender primero su funcionamiento general y sus especificaciones técnicas.

3.2.1. Cómo funcionan

Los sensores LIDAR son dispositivos capaces de medir distancias según el tiempo que tarden los pulsos en ir, rebotar en una superficie (o volumen) y volver al sensor. La medición se realiza calculando el tiempo que tarda el pulso de luz en volver al sensor desde su emisión. En función de su aplicación, estos dispositivos pueden tener uno o varios haces láser, distintos ángulos de percepción y diferentes alcances [8][9].

Para aplicaciones topográficas y de mapeo de relieves, los datos que generan estos sensores se componen generalmente de puntos tridimensionales en el espacio relativo a la posición del sensor. Si el sensor se desplaza (como en el caso de un UAV), el movimiento y rotación del sensor se debe tener en cuenta para el cálculo del punto real. Los datos generados que componen la nube de puntos se guardan normalmente en archivos de formato estandarizado «LAS» [10].

Su uso no sólo se limita a la medición de distancias a superficies, si no que a partir del comportamiento de la refracción de la luz, se puede también usar para mediciones en volúmenes y propiedades de materiales.

Más específicamente, LIDAR encuentra usos de diferentes maneras en muchos sectores, entre ellos destacan:

- Topografía y mapeo de relieves [8].
- Vehículos autónomos y robótica, dónde LIDAR permite generar mapas tridimensionales y otorgar percepción del entorno [11].
- Sensores de velocidad rápidos y de alta precisión [12]
- Óptica adaptativa en la astronomía para eliminar las perturbaciones de la atmósfera en las observaciones desde los observatorios terrestres [13].

- Gestión forestal, dónde el LIDAR permite generar mapas precisos de los materiales combustibles presentes en los suelos para su uso en la lucha contra incendios [14].
- Agricultura, dónde el LIDAR ha ganado popularidad gracias a aplicaciones que permiten medir densidad de población y ratio de crecimiento de las plantas [15], y también se puede usar para estudiar pendientes y detectar insectos.
- Estudio de la atmósfera gracias a que LIDAR permite estudiar gases, nubes y vientos entre otros [16].

3.2.2. Especificaciones técnicas

La competitividad de los sensores LIDAR se puede determinar mediante sus especificaciones técnicas. Los sensores para el mapeo simple de superficies se componen generalmente de las siguientes especificaciones técnicas:

- Alcance: esta es la distancia en la que un sensor es capaz de detectar un objeto y generar puntos sobre este. En el espacio se usan sensores con varios kilómetros de distancia, pero para usos más comerciales como los vehículos autónomos normalmente no se alcanza el kilómetro.
- Frecuencia de puntos: es la cantidad de puntos por segundo que el sensor es capaz de producir. Actualmente pueden superar el millón.
- Campo de visión: determina los ángulos de visión que el sensor escanea. Está el ángulo vertical y el horizontal que juntos determinan el plano de visión. Existen rangos para todo tipo de aplicaciones [17].
- Patrón de escaneo: determina de que manera se comportan los rayos de luz y por ende cómo se generan los puntos. La mayoría de patrones se pueden determinar mediante dos parámetros:
  - El número de canales, que son el número de líneas de escaneo (generalmente horizontales) que se emiten en el campo de visión.
  - Frecuencia de escaneo, que determina el número de ciclos completos que genera cada segundo. Cuanta más frecuencia, menos cantidad de puntos por ciclo.
- Precisión y exactitud: determinan el posible margen de error de los puntos generados. Actualmente la mayoría están en el orden de unos pocos centímetros.

3.2.3. Progreso, estado del arte y expectativas

La tecnología LIDAR ha visto un progreso muy continuado desde su invención en los años 60. Desde entonces ha mejorado permitiendo reducir costes desde los cientos de miles de dólares, hasta apenas unos cientos para ciertos modelos de hoy en día.

El primer láser operativo fue usado en 1960 [18] tras confrontaciones sobre la invención que llevaron a una demanda de más de veinte años sobre la patente [19]. Poco después, el primer sistema tipo LIDAR apareció en el año 1961 bajo el nombre de «COLIDAR» como un nuevo tipo de radar con la finalidad original de rastrear la posición de satélites [20].

Uno de los primeros usos reales de la tecnología fue en meteorología para mediciones de nubes y contaminación [21], y más notablemente, su uso en la misión del Apollo 15 de la NASA (1971) dónde se usó un instrumento LIDAR para medir la topografía de la superficie lunar [22].

A pesar del potencial de la tecnología, su uso en vehículos en movimiento estuvo muy limitado debido a la inexistencia de sensores de posición global precisos (como GPS), limitando su uso a UAVs grandes de alto coste. Esto fue así hasta la aparición de IMUs de mayor precisión a mediados de los años 80 [23], y en la actualidad ha mejorado gracias a alternativas de bajo coste como la tecnología MonoSLAM [24] que permite aproximar el movimiento del UAV usando una cámara RGB en ciertos entornos.

Para el sector de la topografía, la tecnología LIDAR evolucionó a un ritmo rápido, alcanzando en los años 90 más de 10.000 pulsos por segundo permitiendo mapeo topográfico más completo desde UAVs, y en la actualidad, supera el millón de puntos por segundo [23].

Debido al interés de varios sectores, se espera que el crecimiento de esta tecnología no se detenga en el futuro próximo reduciendo costes y mejorando precisión, rango, consumo, etc. alcanzando nuevos usos con mayor accesibilidad.

Modelos comerciales actuales

En esta sección se mencionan algunos modelos reales estado del arte y sus características como referencia para el posterior diseño de los sensores en simulador.

Actualmente compañías como Velodyne y SICK comercializan sensores lidar para el escaneo de superficies. Estos sensores pueden alcanzar los millones de puntos por segundo y estar preparados para la conducción autónoma a costes del orden de los mil dólares. A continuación se muestran algunos ejemplos de modelos estado del arte para diferentes finalidades:

- El modelo NEPTEC Opal 3 P1000 (Panoramic) que posee un alcance de hasta un kilómetro de distancia con una precisión de 3.5cm y un peso de 13kg [17]. Este modelo se ha desarrollado con finalidades de detección de drones y seguridad desde tierra.

- Por otro lado, se están desarrollando alternativas más ligeras, pequeñas y de menor coste compatibles con su uso en drones como el sensor HDL-32E de Velodyne, que, a 1kg de peso, tiene un alcance de 100m con una precisión de 2cm y más de 1 millón de puntos por segundo [25].

- También se comercializan modelos de drones completos diseñados para el uso de LIDAR, como el modelo Zenmuse L1 (dron completo) de la compañía DJI, que tiene un alcance de hasta 100m con una precisión de 5-10cm a 240.000 puntos por segundo y hasta 2km de cobertura de vuelo que además lleva una cámara RGB que permite la combinación de LIDAR con fotogrametría [26].

3.3. Navegación autónoma de UAVs

Según el contexto de uso, el problema de la navegación autónoma de UAVs se puede exponer a varios retos. Desde navegación en línea recta de un punto A a un punto B, hasta carreras de drones autónomos de alta velocidad [27], existe una variedad de problemas que requieren enfoques distintos.

En esta sección se ponen ejemplos de algunas soluciones para problemas de interés no triviales en la navegación autónoma de drones. Esto se hace con la finalidad de obtener conocimiento útil para el diseño del algoritmo de navegación de este proyecto. Estos son problemas complejos que no están completamente resueltos y en muchos casos se usa aprendizaje automático.

3.3.1. Planificación y navegación de rutas

La planificación y ejecución autónoma de trayectorias de navegación es uno de los problemas más importantes y recurrentes en el área de la navegación autónoma. Este problema puede involucrar retos como minimizar el tiempo de viaje, evitar obstáculos y explorar el entorno.

Actualmente existen soluciones con UAVs capaces de desarrollar trayectorias optimizadas en entornos desconocidos, evitar obstáculos en tiempo real e incluso generar mapas detallados del entorno.

En [28] [29] se muestra un algoritmo capaz de generar una ruta entre varios puntos en un entorno desconocido. Este enfoque usa el algoritmo D*-Lite y algoritmos genéticos en varias partes. El vehículo genera primero un mapa del entorno mediante exploración y lo usa para producir un camino general navegable. Posteriormente, los

vehículos navegan el camino general usando sensores LIDAR para evitar posibles obstáculos (navegación local).

Alternativamente, usando aprendizaje por refuerzo, en [30] consiguen que el vehículo siga a un objetivo manteniendo una trayectoria "suave" y evitando obstáculos en tiempo real. Para esto se entrenó completamente un agente usando aprendizaje por refuerzo con el algoritmo DDPG en redes LSTM.

Contemplando los resultados recopilados en la survey [31], para el control de drones mediante aprendizaje por refuerzo destacan especialmente los algoritmos DQN, PPO y TRPO.

3.3.2. Exploración de entornos

La exploración autónoma de entornos es de interés para operaciones de rescate dónde los UAVs pueden rastrear una zona rápidamente y localizar víctimas de forma más eficiente y que requiere menos recursos. Actualmente existen soluciones capaces de generar mapas de

En [7] se usa la técnica LSD-SLAM (ver Sección 3.1) para generar un mapa interno tridimensional del entorno basado en árboles octales y dónde la navegación ocurre usando principios matemáticos para intentar maximizar el área nueva observada.

En [32], se usa aprendizaje por refuerzo profundo para resolver la tarea de exploración de un entorno con la finalidad de encontrar objetivos. Se usan mapas bidimensionales que registran la trayectoria del vehículo y la posición de obstáculos para facilitar una exploración eficiente. La arquitectura de DRL toma como input una imagen blanco y negro que captura el dron, además del mapa local, y usa los algoritmos DQN, DDQN y DRQN para el aprendizaje.

3.3.3. Conclusiones

El aprendizaje automático es un recurso muy usado para resolver problemas complejos de navegación de UAVs [31]. Esto ocurre probablemente porque estas tareas requieren un nivel de adaptabilidad y precisión complicado de obtener mediante algoritmos más convencionales y además en muchos casos se usa visión artificial, área en la que actualmente destaca el aprendizaje automático.

En cuanto a la exploración de entornos, se suelen usar representaciones auxiliares del entorno generados mediante observación como "OctoMaps" [33]. En los casos en los que se usa aprendizaje por refuerzo, así como arquitecturas recurrentes de aprendizaje por refuerzo que le otorgan memoria al agente y con ello potencialmente percepción de su posición en el entorno.

4. APRENDIZAJE POR REFUERZO

En este capítulo se realiza un breve estudio de los elementos más relevantes en aprendizaje por refuerzo así como una taxonomía y una comparación de algunos algoritmos estado del arte para problemas de aprendizaje por refuerzo.

La finalidad de este capítulo es hacer una rápida investigación general del área para poder diseñar y argumentar el algoritmo de navegación autónoma del dron.

4.1. Descripción

El aprendizaje por refuerzo (Reinforcement Learning, RL) es el área del aprendizaje automático que estudia la toma de acciones de agentes inteligentes en base a algún tipo de recompensa del entorno a maximizar. La finalidad de sus métodos es obtener agentes capaces de tomar acciones para encontrar y alcanzar estados que maximicen un valor de recompensa del entorno [34].

El aprendizaje por refuerzo es uno de los tres paradigmas principales del aprendizaje automático junto con aprendizaje supervisado y aprendizaje no supervisado, existiendo diferencias fundamentales entre el aprendizaje supervisado y el aprendizaje por refuerzo. En aprendizaje supervisado se conocen las salidas deseadas del algoritmo, lo que permite extraer una métrica de error basada en la diferencia de la salida del algoritmo con la ya conocida, lo que finalmente permite aplicar eficientes algoritmos como descenso de gradiente. En aprendizaje por refuerzo sin embargo, no es posible determinar cuál es *la mejor acción* y por lo tanto y asumiendo que las acciones no son reversibles, los algoritmos deben aprender a base de prueba y error intentando maximizar el valor de recompensa del entorno.

Cabe destacar que la naturaleza del aprendizaje por refuerzo recoge inspiración en la psicología conductista [35]. De hecho, el término «refuerzo» proviene originalmente de la psicología refiriéndose al proceso por el que un organismo puede responder a un estímulo del entorno y ver su comportamiento alterado temporal o permanentemente frente a este estímulo. Un ejemplo de refuerzo en el aprendizaje de ratas es la tendencia de estas a aprender hábitos para obtener alimentos específicos [36]. En este caso el alimento es un estímulo y los hábitos son las acciones aprendidas como esfuerzo para poder obtener este estímulo de forma consistente.

El aprendizaje por refuerzo en la inteligencia artificial comparte similitudes con el refuerzo en la psicología conductista. Se puede considerar que tanto en el aprendizaje por refuerzo como en la psicología conductista, el concepto de recompensa es una guía general del aprendizaje de un organismo o agente que pueden desarrollar comportamientos complejos a partir de esta.

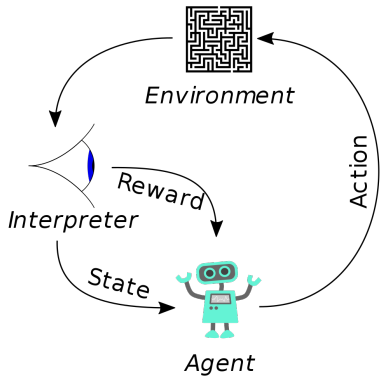


Figura 4.1: Flujo de información en escenarios convencionales en RL [66].

En general la estructura de los escenarios de aprendizaje por refuerzo se puede dividir en dos partes: entorno y agente, dónde el agente recibe observaciones y una recompensa del entorno y a partir de ello toma acciones y aprende para maximizar la recompensa que recibe. En muchos casos se puede considerar que la recompensa y las observaciones provienen de un tercer elemento intérprete que digiere la información abstracta del entorno para el agente e indica la recompensa correspondiente según el estado del entorno. Véase Figura 4.1.

4.1.1. Entorno y recompensa

El entorno es el proceso que ofrece unas observaciones y un valor de recompensa en función de su estado interno. Las observaciones definen completa o parcialmente el estado del entorno. El estado del entorno cambia en función de las acciones realizadas por el o los agentes y, potencialmente, en función del tiempo [37].

Los entornos pueden ser: (1) deterministas o estocásticos en función de si su estado cambia siempre de la misma manera en las mismas circunstancias o por el contrario, si hay factores probabilísticos; (2) totalmente o parcialmente observables en función de si las observaciones otorgadas definen completamente el estado del entorno o al contrario, si hay información del entorno que no se está otorgando; (3) discretos o continuos en función de si las acciones se toman en momentos discretos de tiempo o si se pueden tomar en cualquier momento de un intervalo continuo de tiempo; (4) finitos o infinitos según si el tamaño del conjunto de sus estados es finito o infinito.

Proceso de decisión de Markov

Uno de los conceptos más recurrentes en el aprendizaje por refuerzo son los procesos de decisión de Markov (Markov Decision Process, MDP), una extensión de las cadenas de Markov. Los MDPs son un modelo matemático que permite definir un entorno como un proceso de toma de decisiones definido por el conjunto de todos

Notación	Descripción	Tipo
s, s'	Estado del entorno	Vector de valores
S	Conjunto de todos los estados	Conjunto
a	Acción	Vector de valores
A	Conjunto de todas las acciones	Conjunto
r	Recompensa	Valor escalar
t	Momento temporal discreto	Valor escalar

Cuadro 4.1: Notación de elementos en un entorno de RL.

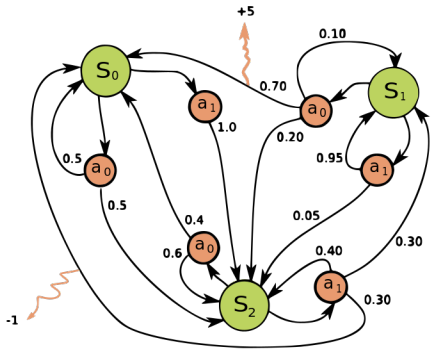


Figura 4.2: Ejemplo de MDP [67].

los estados posibles S tal que en un momento concreto el entorno se encuentra en uno de los estados y se pueden tomar decisiones para provocar una transición de estados [37].

En general, un MDP se puede definir como una 4-tupla tal que (S, A, P_a, R_a) [38] donde:

- S es el conjunto de todos los estados posibles en el entorno.
- A es el conjunto de todas las acciones posibles.
- P_a es la función de probabilidad tal que la probabilidad de transicionar de un estado s a otro s' bajo la acción a está definida por $P_a(s, s')$.
- R_a es la función de recompensa tal que para una transición de un estado s a s' bajo la acción a, la recompensa está definida por $R_a(s, s')$.

De esta manera, una instancia de un entorno en un momento cualquiera t se puede definir como su estado $s \in S$ y se puede tomar una acción $a \in A$ para provocar una transición al siguiente estado $s' \in S$ en el momento $t + 1$ con una probabilidad $P_a(s, s')$. La recompensa dada por esta transición se define en $R_a(s, s')$.

Esto permite definir todos los elementos generalmente necesarios de un entorno de aprendizaje por refuerzo. En el proceso de ejecución de un agente, para un mo-

mento t, el agente extrae las observaciones del estado s, produce una acción a que provoca una transición a un estado s', y recupera una recompensa $r = R_a(s, s')$.

Para funcionar, los MDPs asumen que los conjuntos de estados y acciones son discretos, finitos y los estados definen por completo el entorno. Es por esto que no todos los entornos pueden definirse mediante MDPs simples, sino que según la naturaleza del entorno, se requieren variaciones que permitan contemplar otros casos:

- Proceso de decisión de Markov parcialmente observable (POMDP) [39]. Esta es una extensión de las MDPs que permite modelar entornos cuyos estados son incompletos y no tienen toda la información necesaria para determinar transiciones con seguridad. Definida por la 7-tupla $(S, A, T, R_a, \Omega, O, \gamma)$, permite modelar posibles transiciones bajo estados incompletos. Un ejemplo de POMDP es el juego de póker donde se tienen que tomar acciones sin conocer las cartas de los oponentes.
- Proceso de decisión de Markov para tiempo continuo. Definido por los mismos 4 elementos que los MDPs, adapta estos para permitir modelar conjuntos continuos (e infinitos) de estados y acciones, y con ello sus transiciones y recompensas.

En todos estos casos la información que utiliza el agente es (generalmente) en esencia la misma: un conjunto de estados observables S, un conjunto de acciones realizables A y una recompensa relacionada con cada transición $R_a(s, s')$.

Recompensa

La recompensa (r) es un valor generado junto a una transición de estados dada una acción, y que, en MDPs simples, siempre es el mismo valor para la misma transición. La recompensa generalmente representa el rendimiento de las acciones tomadas, de tal manera que cuanto mayor sea, mejor rendimiento [37].

La función de recompensa $R_a(s, s')$ en un MDP es la función que mapea cada posible transición con un valor de recompensa tal que $r_t = R_a(s, s')$, donde en el momento t se ha tomado la acción a y ha resultado en una transición del estado s a s' generando con ello el valor de recompensa r_t.

El objetivo de un agente en aprendizaje por refuerzo es maximizar el valor de recompensa acumulado en el tiempo. Es por esto que la recompensa es la guía del agente y debe representar adecuadamente los objetivos de diseño o de otra manera puede llevar a soluciones indeseadas.

Una buena recompensa debe otorgarse consistentemente en todo el espacio de estados y evitando saltos de valores muy grandes exceptuando para penalizaciones

bruscas. Casos de malas recompensas pueden llevar al "efecto cobra", que es cuando los agentes explotan la función de recompensa sin alcanzar necesariamente el objetivo, o también pueden provocar estancamientos o entrenamientos muy lentos [40].

4.1.2. Agente

En inteligencia artificial, un agente inteligente es una entidad autónoma capaz de tomar acciones a partir de las observaciones de un entorno bajo una finalidad. Aunque en este caso se tratan de agentes en software, la definición se puede llevar a mecanismos simples como el sistema de enfriamiento de una nevera, que "observa" la temperatura del interior de la nevera y reacciona aumentando o reduciendo el sistema de enfriamiento.

En aprendizaje por refuerzo el agente responde al entorno y aprende a partir de experiencia, siendo este aprendizaje generalmente por el valor de recompensa del entorno. De esta manera, el agente intenta aprender a tomar acciones que le acerquen a mejores valores de recompensa a corto o largo plazo [34].

Política del agente

La lógica que procesa las observaciones y determina la mejor acción se conoce como la «política», que se suele representar con el símbolo π y es aprendida. Generalmente, la política se define como una función que, dado un estado y una acción, devuelve un valor que representa la probabilidad de realizar esa acción en el estado dado:

$$\pi : A \times S \to [0, 1] \tag{4.1}$$

De esta manera, el valor resultante representa en cierta forma la recompensa esperada, donde la acción con mayor valor es la que espera mayor recompensa.

La meta del aprendizaje por refuerzo es aprender una política que maximice el valor de recompensa mediante algoritmos de aprendizaje basados en la experiencia.

Función de valor y valores Q

La función de valor, normalmente aprendida por el agente durante el entrenamiento, representa la expectativa de recompensa a largo plazo (conocido como retorno, R) dada la mejor trayectoria conocida por la política del agente a partir del estado en el que se encuentre [37]. Formalmente:

$$V_\pi(s) = E[R]|R = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + ... = \sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} \tag{4.2}$$

Donde $V_\pi(s)$ es la función de valor para la política π, $E[R]$ es el retorno esperado, r_t es el valor de recompensa en el momento t y γ es el valor de descuento.

Muchos algoritmos estado del arte basan su funcionamiento en modelar esta función para guiar la política a largo plazo. Esta es la base de Q-Learning (y sus variaciones), que modelan la función de valor mediante los *valores Q*.

Los valores Q son aquellos que representan las expectativas de retorno dada una acción a en un estado s [41]. La función de valores Q, $Q(s, a)$, que modela los valores Q para las transiciones de un MDP, es similar a la función de valor en que estima un retorno, pero en este caso para cada acción dado un estado. Asumiendo que la política tome las acciones con mayor valor Q, entonces, en un algoritmo de Q-Learning, la función de valor se puede interpretar como:

$$V_\pi(s) = max_i(Q_\pi(s, a_i)) \tag{4.3}$$

Para todas las acciones a_i disponibles en el estado s. Esto significa que la función de valor para un estado s es equivalente al valor Q de la mejor acción en ese estado.

4.2. Clases de aprendizaje por refuerzo

4.2.1. Model-Based vs Model free

En el aprendizaje por refuerzo existen una variedad de algoritmos con funcionamientos muy distintos y con pros y contras muy contrastados. Es común que en problemas de aprendizaje por refuerzo se deba probar experimentalmente el rendimiento de distintos algoritmos antes de conocer la mejor alternativa.

Primero, en aprendizaje por refuerzo todos los algoritmos se pueden clasificar bajo un primer criterio [42]:

- Si dependen directamente de la distribución de recompensa del entorno según el estado, ya sea directamente proveído de forma externa o figurado por el agente; se denominan algoritmos «model-based».
- Por el contrario, si su aprendizaje modela otro tipo de heurística o reglas para guiar las acciones, se denominan algoritmos «model-free».

Teniendo en cuenta el tipo de problema relevante en este trabajo y las tendencias actuales en el sector, en esta sección se hablará únicamente de algoritmos de tipo **model-free**.

4.2.2. Función de valor vs política

Dentro de los algoritmos de tipo model-free se pueden distinguir principalmente dos clases: basadas en función de valor, y basadas en búsqueda directa de la política. En esta sección las basadas en función de valor se dividen en métodos de Monte Carlo y temporal difference learning.

También se habla de una tercera clase de algoritmos que combinan función de valor y política simultáneamente. Estos algoritmos suelen modelar la función de valor y la política por separado y se conocen como algoritmos actor-critic.

Métodos de Monte Carlo (función de valor)

En términos generales, los métodos de Monte Carlo son algoritmos basados en repetido muestreo aleatorio de algún tipo para extraer resultados experimentales [43]. En el caso de aprendizaje por refuerzo, para problemas con espacio de estados y acciones discretos, el muestreo aleatorio se aplica en las acciones con la finalidad de obtener experiencia en todos (o parte de) los estados y poder aproximar así la función de valor de transiciones $Q^*(s, a)$ de forma experimental [44]. De esta manera Monte Carlo es uno de los métodos más simples de aprendizaje por refuerzo que además explora todo el espacio de estados.

La ventaja de Monte Carlo es que funciona muy bien en MDPs estocásticos y POMPDs (entornos donde no toda la información es conocida) con número de estados reducido, pues extrae una distribución de recompensas experimentalmente y con ello señala cuáles son los estados con mayores probabilidades de ofrecer mejor recompensa.

La principal desventaja de esta metodología es que requiere observar todos los estados del entorno varias veces lo cual se vuelve inviable muy rápido en problemas con espacios de estados más grandes. También puede ser problemático cuando el muestreo de estados es muy irregular haciendo que rara vez se alcancen algunos estados, requiriendo así mucho más muestreo para completar la distribución o acabar con una distribución incompleta.

Temporal difference learning (función de valor)

Temporal difference learning (TDL; en español, métodos de diferencias temporales) es actualmente una de la técnicas más populares en aprendizaje por refuerzo y es usada por muchos algoritmos de estado del arte gracias a su versatilidad. Al igual que los métodos de Monte Carlo, TDL aprende mediante la aproximación de una función de valor, pero en lugar de usar aleatoriedad en la toma de acciones, usa su propia política basada en expectativas de recompensa a corto y largo plazo. De esta manera puede aprovechar lo ya aprendido para aprender más rápido [41].

La clave de estos métodos es que intentan modelar la función de valor de estado a partir de lo que se conoce como el "retorno descontado" (discounted return), que es básicamente proporcional a la recompensa acumulada que espera alcanzar a partir de la mejor trayectoria de acciones que la política conoce. Matemáticamente, el retorno descontado de una trayectoria se puede modelar como:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \tag{4.4}$$

Donde $\gamma \in (0, 1]$ es el factor de descuento y r_i son las recompensas alcanzadas tras ejecutar las acciones dadas por la política.

El algoritmo más notorio en esta clase es Q-Learning y sirve como base de muchos otros algoritmos usados en la actualidad como DQN o DDPG. Q-Learning modela la función de valor mediante los valores Q almacenando uno distinto para cada par acción-estado del MDP, es decir, es de tipo «tabular» [41]. Este algoritmo aprende estas expectativas de recompensa en cada valor Q influido por dos partes: el valor de recompensa del entorno en ese estado (recompensa inmediata) y el valor de estado aprendido de la mejor transición siguiente (recompensa futura):

$$Q'(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot max_a Q(s_t + 1, a) - Q(s_t, a_t)) \tag{4.5}$$

Donde r_t es el valor de recompensa al transicionar al estado s_{t+1} y la expresión siguiente $\gamma \cdot max_a Q(s_{t+1}, a)$ representa la estimación descontada del valor futuro óptimo. De esta manera los valores de estado futuros se "propagan" a los anteriores según el factor de descuento.

Las acciones a realizar se determinan según una probabilidad definida por un factor de exploración $\epsilon \in [0, 1]$ que determina si tomar la acción dada por la política o una aleatoria con fines exploratorios. Es importante tener un factor de exploración mayor de 0 o de otra manera el algoritmo convergerá prematuramente sin explorar bien el espacio de estados.

Q-Learning más que un sólo algoritmo, ha define una clase de algoritmos que se basan en esta ecuación para aprender sus valores.

En general, la ventaja de estos algoritmos es que permiten converger sin necesitar explorar meticulosamente todo el espacio de estados, pero puede implicar que no se alcance la política óptima o incluso prevenir su convergencia completamente. A pesar de esto, actualmente es uno de los enfoques por excelencia para tratar MDPs y POMDPs con espacios de estados de gran tamaño.

Búsqueda directa de la política

A pesar de su probada efectividad, las aplicaciones basadas en función de valor tienen sus desventajas como su falta de robustez frente a las irregularidades en los valores de recompensa de los POMPD que puede provocar convergencia a políticas subóptimas [45].

Una alternativa que ha demostrado funcionar mejor en muchos casos es aproximar directamente la política de toma de acciones sin necesidad de estimar ninguna función de valor [46]. Esta política, como todas, es una transformación del espacio de estado al espacio de acciones, siendo la diferencia que la función se aprende directa y completamente sin necesidad de estimar previamente expectativas de recompensa [50].

Un ejemplo de algoritmo de esta clase es REINFORCE [51] que utiliza ascenso de gradiente para tratar de clasificar las mejores acciones según el estado intentando maximizar el «retorno de la trayectoria» que es esencialmente el sumatorio de la recompensa obtenida en una secuencia de acciones. Se trata de *ascenso* de gradiente y no *descenso* puesto que se intenta maximizar el retorno en lugar de minimizar un error. Formalmente:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta U(\theta_t) \tag{4.6}$$

Donde θ son los parámetros a actualizar, α es el ratio de aprendizaje y $U(\theta)$ es el retorno de recompensa esperado que se busca maximizar.

Aunque REINFORCE (inventado en 1992) es un algoritmo anticuado, existen actualmente alternativas estado del arte que compiten con los algoritmos basados en función de valor.

Estos métodos permiten centrar el aprendizaje más en la toma de acciones sin tener que modelar la recompensa del entorno lo cual ha demostrado ser útil en muchos casos, especialmente cuando se trata de POMPDs con grandes espacios de estados donde puede ser más complicado modelar la función de valor [46]. A pesar de esto, los métodos basados en función de valor siguen demostrando mejores resultados en la mayoría de problemas de aprendizaje por refuerzo.

Algoritmos actor-critic

Los algoritmos actor-critic son una clase de algoritmos que combinan función de valor con política separada. En estos casos la función de valor se modela directamente a partir del entorno, y luego, en lugar de aprender la política a partir de las trayectorias tomadas, la aprende a partir de la función de valor modelada [47].

La política en estos algoritmos normalmente está completamente separada de

la función de valor y normalmente selecciona las acciones a partir de sí mismo. Es entonces donde entra la función de valor que "critica" las acciones tomadas por la política y aprende intentando maximizar la función de valor mientras simultáneamente la función de valor intenta aprender del entorno.

Actualmente, especialmente desde la aparición del algoritmo PPO, muchos de los algoritmos más populares se basan en esta clase de aprendizaje.

4.3. Deep Reinforcement Learning

El Deep Learning o aprendizaje profundo es un conjunto de algoritmos de aprendizaje automático que normalmente hacen uso de redes neuronales artificiales o similares permitiendo enfrentar todo tipo de problemas de clasificación y regresión con efectividad [48]. Este tipo de aprendizaje se ha popularizado mucho últimamente gracias a su capacidad de abstracción, efectividad universal y capacidad de paralelización.

El Deep Reinforcement Learning (DRL) o aprendizaje por refuerzo profundo es la rama del Deep Learning para el aprendizaje por refuerzo. Esta área del aprendizaje por refuerzo destaca pues sus algoritmos son muy versátiles gracias a su capacidad de modelar MDPs y POMDPs de forma abstracta y universal, y con ello son capaces de enfrentarse a problemas muy complejos con algoritmos flexibles [49].

Al ser una rama del Deep Learning, estos algoritmos suelen hacer uso de arquitecturas basadas en redes neuronales profundas. Las redes neuronales profundas son capaces de aproximar todo tipo de funciones gracias a su no linearidad, y es esa propiedad la que se aprovecha para aproximar políticas.

Estos algoritmos además, usando redes neuronales convolucionales, se pueden usar para resolver problemas de aprendizaje por refuerzo que usen visión. Estos algoritmos se pueden usar para modelar políticas directas (como por ejemplo en REINFORCE), funciones de valor (como Deep Q-Learning), o ambas simultáneamente para algoritmos tipo actor-critic (como PPO).

4.4. Estado del arte en RL

A continuación se nombran y explican algunos algoritmos de aprendizaje por refuerzo que han demostrado ofrecer resultados estado del arte en problemas complejos. No hay ningún algoritmo general que funcione bien en todos los entornos si no que según el tipo de problema algunos rendirán mejor mientras que otros podrán ser hasta incompatibles.

También es importante tener en cuenta si los algoritmos están diseñados para espacios de estados y acciones continuos o discretos, y si son compatible con la

ejecución de varios agentes simultáneos (MP) para acelerar el entrenamiento.

Normalmente los algoritmos de Deep Reinforcement Learning al usar redes neuronales artificiales son compatibles con todo tipo de espacio de estados y acciones, aunque hay excepciones.

Un benchmark de los algoritmos estado del arte en DRL se puede encontrar en RL Baselines Zoo [52]. Este benchmark se compone de los diferentes entornos del simulador pybullet y de algunos juegos de Atari.

Deep Q-Networks (DQN)

Estos algoritmos basados en Q-Learning usan redes neuronales para modelar la función de valor usando esencialmente la misma propiedad de aprendizaje que Q-Learning (Ecuación 4.5). En este caso en lugar de actualizar un valor Q en una tabla, el algoritmo altera los parámetros de la red neuronal mediante descenso de gradiente para aproximar este valor Q [53].

Gracias al uso de redes neuronales, este algoritmo obtiene varias ventajas sobre su versión tabular. La ventaja más clara es la capacidad de modelar espacios de estados muy grandes sin requerir tanta memoria. Otra ventaja es que el algoritmo tiene capacidad de generalización, pudiendo reutilizar experiencia en estados nunca observados antes. Actualmente la mayoría de algoritmos estado del arte para problemas complejos usan redes neuronales u otras estructuras que también poseen estas ventajas.

La principal desventaja de las DQNs es que su aprendizaje es considerablemente más lento que otras alternativas, requiriendo más tiempo de computación y además, no suele ofrecer los mejores resultados.

Una variación son las **Double Deep Q-Networks (DDQN)** donde, para determinar la política, se mantiene una copia de la función de valor que se mantiene congelada mientras la original se sigue actualizando. Entonces, cada cierto número de pasos se renueva esta política congelada con el modelo actualizado. Esto mantiene mas "estable" la política y puede demostrar mejores convergencias [54].

Proximal Policy Optimization (PPO)

PPO [55] es un algoritmo basado en los algoritmos A2C y TRPO. Inspirado en A2C, PPO es capaz de aprender a partir de varios agentes simultáneamente, pudiendo así acelerar el aprendizaje [56]; y, inspirado en TRPO, utiliza una técnica conocida como "Trust Region Policy Optimization" (TRPO) que ofrece "guaranteed monotonic improvement" (mejora monótona garantizada) [57], que aunque no es realmente una garantía absoluta en todos los casos, si que ofrece una mayor consistencia en las mejoras reduciendo la frecuencia de caídas de rendimiento.

PPO, siendo un algoritmo de tipo actor-critic, actualiza una política específica durante el entrenamiento. Para esta actualización, este algoritmo introduce el uso de "clipping" para evitar que tras la actualización la política cambie mucho respecto a la anterior, resultando en entrenamientos con mejorías más consistentes. De aquí proviene el nombre de *proximal* policy optimization.

Este algoritmo consigue uno de los mejores rendimientos destacando por su facilidad de implementación y requiriendo poca afinación de hiper-parámetros. Además, consume menos memoria al no requerir almacenar replay buffers que pueden tomar varios gigabytes de memoria.

Sample Efficient Actor-Critic with Experience Replay (ACER)

ACER es un algoritmo tipo actor-critic que implementa una función de valor con replay buffer similar a DQN y la utiliza para optimizar su política introduciendo varias mejoras en la actualización de valores y la arquitectura [58].

Aunque este algoritmo destaca por ofrecer de los mejores resultados en muchos problemas, tiene el inconveniente de ser especialmente complejo y requerir el afinamiento de varios hiperparámetros.

5. DESARROLLO DEL TRABAJO

5.1. Diseño general

Como ya se explica en el Capítulo 2, en este proyecto se busca, haciendo uso de drones, crear un sistema totalmente autónomo capaz de generar mapas tridimensionales basados en nubes de puntos LIDAR.

5.1.1. Por qué LIDAR

En este trabajo se busca crear en simulador un sistema autónomo capaz de generar mapas completamente tridimensionales haciendo uso de drones en varios tipos de entornos, incluyendo entornos cerrados como interiores de edificios. Para esto se necesita una percepción consistente de todas las superficies sólidas que componen el entorno.

LIDAR es capaz de percibir con precisión superficies y modelarlas como nubes de puntos. Esto permite recrear fielmente los sólidos de un entorno con un nivel de detalle tan alto como la precisión del sensor lo permita. Además, como se ha visto previamente, su uso en drones es viable y se esperan mejoras en tamaño y peso en el futuro.

Una de las dos desventajas principales de los sensores LIDAR usados en este caso es que la información que recaba es exclusivamente espacial y no es capaz de recoger datos de materiales, colores u otros datos del entorno. Datos que pueden ser necesarios para varias aplicaciones como estudios geológicos. Aunque este trabajo se limita a generar un modelo exclusivamente espacial, este problema se podría solucionar combinando LIDAR con el uso de cámaras para complementar las superficies con texturas extraídas de las cámaras como es el caso en [1].

La otra desventaja es la sensibilidad al ruido de elementos móviles que pueda percibir. Puesto que aquí se usa simulador, este problema no es existente dado que los entornos se pueden establecer para que sean completamente estáticos, pero en la vida real, en principio sería posible usar técnicas en las que se eliminan las inconsistencias comparándolas en diferentes ciclos de captación de datos.

En general, LIDAR cumple los requisitos principales del objetivo de este trabajo y además destaca por su precisión y simplicidad para el tratamiento de los datos comparado con las técnicas más complejas de fotogrametría.

Siguiendo los intereses de la industria, lo más apropiado sería combinar LIDAR con fotogrametría para obtener mapas tridimensionales completos y texturizados, pero esto se sale del alcance de este trabajo.

5.1.2. Arquitectura general del sistema

El sistema de mapeo autónomo en general se compone de varios elementos. En el nivel más alto tenemos al sistema de navegación autónoma y el entorno. En este caso el entorno es simplemente el simulador que simula las acciones que pueda tomar el sistema de pilotaje autónomo.

Por otro lado, el sistema de navegación autónoma se compone de dos elementos: (1) la herramienta de mapeo de entornos que a través de la información recogida por los sensores del dron, es capaz de procesar y generar eficientemente un mapa pudiendo variar el nivel de detalle, y (2) el sistema de aprendizaje por refuerzo, compuesto a su vez del agente y su entorno abstracto.

En general, el entorno abstracto de aprendizaje por refuerzo engloba la herramienta de mapeo y la comunicación con el simulador para ofrecer una interfaz abstracta al agente. Esta interfaz ofrece de esta manera percepción del entorno, acciones para realizar y un valor de recompensa y función de reinicio para el entrenamiento. El agente entonces puede ser controlado por un humano o por un algoritmo de aprendizaje por refuerzo y con ello es posible generar un mapa del entorno con el detalle deseado solo con mover el dron.

5.2. Simulación de drones: AirSim

Debido a las limitaciones de la tecnología, y a la cantidad de datos necesarios para entrenar un agente por aprendizaje por refuerzo, este proyecto se realiza completamente en simulador.

5.2.1. Necesidades

Para cumplir los objetivos de este trabajo, es necesario un simulador que permita simular drones y sensores LIDAR de forma realista, además de ofrecer una interfaz de programación de aplicaciones que permita entrenar a un agente por aprendizaje por refuerzo.

Por esto, se establecen una serie de requisitos que se espera que el simulador cumpla.

Requisitos del simulador

1. Simulación de drones realista.
2. Simulación de LIDAR.
    a) Posibilidad de montar sensor LIDAR en dron.

   b) Flexibilidad en la configuración de las características del sensor.

3. Interfaz de programación de aplicaciones (API) completa y flexible.

   a) API capaz de ofrecer datos de LIDAR.

   b) API capaz de ofrecer un control completo y síncrono del dron.

   c) API capaz de reiniciar simulaciones con diferentes entornos y posiciones iniciales del dron.

4. Capacidad de simular distintos entornos.

5. Buen rendimiento.

5.2.2. AirSim

AirSim es un simulador de vehículos, especialmente coches y drones, basado en el motor gráfico Unreal Engine capaz de simular distintas configuraciones de vehículos bajo control manual o automatizado. El simulador permite un control informatizado de los vehículos a través de su flexible interfaz de programación de aplicaciones (API) [59].

Gracias a que el simulador está basado en Unreal Engine, es posible establecer la simulación en cualquier entorno 3D que se pueda importar a un proyecto de Unreal Engine (en formato .uproyect), permitiendo incluso crear entornos personalizados con facilidad.

AirSim cumple con los requisitos de los objetivos [Subsección 5.2.1] con bastante efectividad. De hecho, el simulador está en parte diseñado para el entrenamiento de algoritmos de aprendizaje automático [60]. Además, tiene una buena documentación y soporte de la comunidad.

Requisito	¿Lo cumple?
1. Simulación drones.	Sí
2. Simulación LIDAR.	Sí
a) LIDAR en dron.	Sí
b) Configuración sensor.	Sí
3. API Completa	Sí
a) API lidar.	Sí
b) Control completo y síncrono.	Parcial
c) Reinicio simulaciones.	Sí
4. Distintos entornos	Sí.
5. Buen rendimiento	Parcial.

Cuadro 5.1: Cumplimentación de requisitos [Subsección 5.2.1] por AirSim.

AirSim permite simular drones con sensores LIDAR con diversas configuraciones y con la posibilidad de implementar un framework completo para la ejecución de algoritmos de aprendizaje por refuerzo a trvés de su API.

Los únicos problemas son (1) mediante la API, el simulador se comunica con los programas de forma asíncrona, añadiendo dificultades para una recolección de datos robusta; y (2) un rendimiento subóptimo, puesto que Unreal Engine está diseñado para videojuegos con un mayor realismo gráfico (que no es necesario con LIDAR), existen costes computacionales innecesarios para este proyecto donde lo que busca maximizar la velocidad de recolección de datos para entrenar los agentes en menor tiempo. Por suerte, AirSim tiene la ventaja de que permite desactivar el procesado de la imagen para ahorrar recursos.

LIDAR en AirSim

AirSim permite simular el funcionamiento de sensores LIDAR que se pueden montar en los vehículos. La simulación se limita a la emisión de pulsos desde un punto inmaterial y no se simula la estructura del sensor ni nada similar. En este proyecto, los puntos de emisión de pulsos LIDAR se colocan cercanos al dron y se desplazan juntos. Además, la precisión de estos sensores simulados es perfecta y no se da la opción nativa de simular error.

El simulador permite establecer varias configuraciones de sensores LIDAR, pudiendo cambiar su rango, campo de visión vertical y horizontal, puntos por segundo y frecuencia de ciclos. En este proyecto se configuran dos sensores: uno arriba del dron y otro abajo, de tal manera que forman una cúpula de 360^{o} casi completa.

Los datos se recogen en forma de puntos 3D relativos a la posición del sensor. Estos datos van acompañados de información de la posición global y rotación del vehículo o, según como se haya configurado, del propio sensor del que son producidos, permitiendo así calcular las posiciones globales de los puntos.

5.2.3. La API de AirSim

La API es la interfaz que comunica el simulador con los programas diseñados por el usuario. Esta API permite recolectar información del entorno y los vehículos, enviar instrucciones de control para los vehículos simulados y alterar el entorno de varias maneras.

La comunicación de la API con los programas es asíncrono, es decir, que el programa y el simulador se ejecutan por separado y, en el caso de AirSim, se comunican entre sí a través de un puerto IP usando el protocolo de la librería «msgpack-rpc» [61]. Aunque esto permite comunicación entre redes de varios ordenadores, en este caso se realiza de manera local (al mismo ordenador).

Recolección de datos

AirSim permite la completa recolección de los datos de todos los sensores equipados de todos los vehículos instanciados y además permite recolectar la información de las imágenes que genera la ventana gráfica («viewport») y otros datos del estado de los vehículos y el entorno como la posición del vehículo, si ha colisionado, etc.

Para este proyecto sólo se usan los datos del sensor LIDAR, la posición global de los sensores y si el vehículo ha colisionado. Nótese que en el mundo real sería necesario un método de localización de la posición del vehículo mediante GPS y/o sensores cinemáticos.

Control de vehículos

Es posible enviar instrucciones al simulador para controlar el ó los vehículos. Para el caso de los drones multirotor, se puede ordenar su movimiento mediante ángulo de posición (alterando su ángulo de elevación), vector de velocidad y posición de destino.

El control de los vehículos es siempre asíncrono y se basa en rutinas que requieren el uso de mecanismos de sincronización.

Control del entorno

Es posible también alterar factores como el viento, el clima y el momento del día. En este proyecto no se utilizan.

Sincronización y control general del programa

Finalmente, existen instrucciones para el control del simulador desde el programa. Estas instrucciones permiten manejar la conexión con el simulador, reiniciar la simulación, pausarla, etc.

5.2.4. Configuración y uso de AirSim en el proyecto

AirSim está diseñado para permitir varios tipos de uso y con un amplio abanico de configuraciones. En este proyecto se establece una configuración para dar obtener las funcionalidades deseadas, hacer uso de un vehículo con características alineadas a los objetivos de este proyecto, y priorizando rendimiento.

Usando la configuración para vehículos "multirotor" (drones), se crea un dron con dos sensores LIDAR, uno arriba y uno abajo que cubren casi una cúpula completa de 360º. A estos sensores se les pone un rango reducido (alrededor de 10-15 metros), y de 20000 puntos por segundo cada uno a 10 rotaciones/segundo en 32 canales.

Para minimizar el tiempo de entrenamiento se aumenta el valor de "ClockSpeed" a 4, lo que provoca que el simulador intente funcionar a 4 veces la velocidad normal (aunque en este caso nunca lo alcanza) y se deshabilita el display (no se renderiza el entorno) para potenciar el rendimiento.

De cara al entrenamiento AirSim se mantiene ejecutando constantemente mientras el entorno programado usa la API para controlar la simulación. El entorno también tiene la capacidad de cerrar y abrir el proceso para cambiar mapas si es necesario.

5.3. Tratamiento de datos y visualización

Los sensores LIDAR actuales son capaces de generar más de un millón de puntos por segundo que, asumiendo un tamaño de 12 Bytes por punto, se traduce a un flujo de más de 10MB/s. Esto, si se deja ejecutar sin restricciones durante 5 minutos, llenaría un archivo de 3GB de datos que, aunque un ordenador podría procesar, el microprocesador de un dron quizás no. Otro reto consistiría en minimizar el tiempo de ejecución del procesado de datos para acelerar el entrenamiento del agente y reducir los requerimientos de computación del dron.

Por esto, esta sección distinguen tres requisitos:

- Almacenar los datos en una estructura de datos eficiente que permita también la visualización del mapa.
- Reducir la cantidad de datos almacenados. Esto se puede hacer a partir del hecho de que gran parte de los haces LIDAR acaban aterrizando en las mismas áreas mientras el dron se desplaza, provocando que gran parte de los puntos sean redundantes y se puedan omitir.
- Minimizar el coste temporal computacional del algoritmo de tratamiento de datos. El objetivo es permitir que se puedan procesar en tiempo real decenas de miles ó cientos de miles de puntos.

También será necesario disponer de una herramienta que esté preparada visualizar los mapas tridimensionales compuestos de varios megabytes o incluso gigabytes de datos.

5.3.1. OctoMap

Una de las mejores soluciones para la generación de mapas tridimensionales en tiempo real es el framework OctoMap [62] basado en las estructuras de datos «octrees» (o árboles octales). Este framework modela el mapa a partir de haces (como los lanzados por LIDAR) y representa el espacio en vóxels con un tiempo de

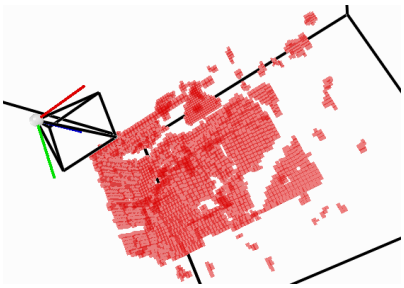


Figura 5.1: Ejemplo de uso de OctoMap en mapeo tridimensional.

inserción y búsqueda de datos de $O(log(n))$. Una de las principales características de este framework es que es capaz de almacenar y representar volumen ocupado, libre y desconocido en un mismo mapa.

La principal ventaja de OctoMap es su eficiencia para procesar los datos. Su algoritmo de inserción permite insertar rápidamente puntos y procesar . Otra ventaja de los árboles octales es que se pueden podar en ciertos casos para ahorrar espacio sin perder información. La implementación usada incluye además un método para el trazado de rayos que se utiliza en este proyecto para generar imágenes que servirán de percepción al agente.

Visualización

Aunque el framework incluye una herramienta de visualización, por temas de compatibilidad en este proyecto se diseña una herramienta nueva para visualizar los mapas. La implementación simplemente extrae los vóxels únicos y los representa como cubos en un framework 3D, en este caso usando Mayavi.

Uso en el proyecto

En este proyecto se usa su implementación de código abierto para almacenar una representación del entorno que el agente explora. La nube de puntos generada por los sensores LIDAR se insertan en esta estructura de datos que se mantiene ligera para permitir los cálculos necesarios en un tiempo bajo.

Puesto que se necesitan cálculos rápidos sobre el mapa para generar la percepción del agente y la recompensa, OctoMap se usará para crear un mapa temporal a una resolución relativamente baja mientras que paralelamente se puede almacenar la nube de puntos completa, que luego será la que se use para generar una representación final del entorno.

5.4. Navegación autónoma mediante aprendizaje por refuerzo

5.4.1. Entorno

Como ya se ha comentado en Subsección 4.1.1, un entorno en aprendizaje por refuerzo tiene que proveer unas observaciones, recompensa, condición de terminación y ejecución de las acciones especificadas por el agente. Junto a esto, debe de poseer también la funcionalidad de reiniciarse para permitir al agente entrenarse de forma fluida.

El entorno general es simulado por AirSim y se requiere implementar una interfaz que extraiga y procese los datos generados para el funcionamiento y entrenamiento del agente. Esta interfaz debe satisfacer los requisitos de generalización «Env» del framework Gym (explicado a continuación).

A continuación se discuten muchas decisiones de diseño para cada elemento del entorno (diseño de recompensa en Subsección 5.4.2).

Percepción del entorno

El objetivo del agente es generar un mapa lo más completo posible del entorno. Para esto se busca que la percepción provea suficiente información cumpliendo ciertos requisitos:

- La percepción debe proveer cierto grado de información sobre la posición del dron y los posibles obstáculos que debe evitar.
- El agente debe tener alguna constancia de las áreas exploradas y por explorar y deseablemente esto se hará a partir de la percepción para evitar forzar estrategias más abstractas o uso de memoria a largo plazo.

Además esta información debe ofrecer deseablemente buena compatibilidad con los algoritmos usados para evitar. Esto es por ejemplo: imágenes de la visión del dron han demostrado funcionar bien como percepción en aprendizaje por refuerzo en Deep Reinfocement Learning [32], mientras que por ejemplo valores abstractos de posiciones en un mapa pueden requerir operaciones más complicadas a las que sea improbable converger adecuadamente.

En este caso al agente se proveen dos estructuras diferentes como percepción (ver Figura 5.2). La primera es una imagen de visión de 360º horizontalmente generada a partir del mapa tridimensional generado con OctoMap usando trazado de rayos. Esta imagen entonces posee un sólo canal (escala de grises) que es en esencia una imagen de profundidad del mapa descubierto alrededor del dron. Esta imagen entonces ofrece principalmente información de los obstáculos al dron para que los pueda evitar, pero

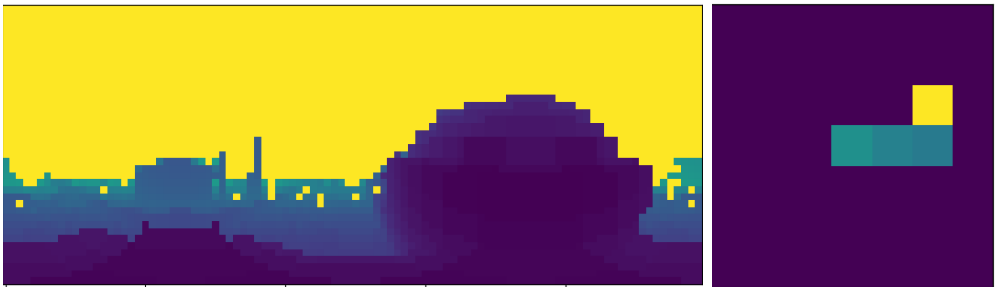


Figura 5.2: Percepción del entorno para el agente. Izquierda: visión del mapa interno basada en profundidad sobre la posición del dron. Derecha: mapa de cuadrantes visitados.

además tiene el potencial de ofrecer pistas al agente de zonas que debería explorar (como por ejemplo detrás de una pared).

La otra estructura es un mapa general del entorno a explorar de 5x5 casillas con valores entre 0 y 1 que se transforma a un vector plano de dimensión 25. El mapa representa la cantidad de pasos que el dron ha estado en cada cuadrante del mapa donde un valor de 0.5 indica la casilla más visitada y 0 indica casillas sin visitar. Entonces el valor 1 se usa sólo para la casilla en la que el dron se encuentra en el momento. Esta estrategia de representar varios datos en el mismo mapa de canal único se usa también en [32] con buenos resultados.

Se valora también utilizar una variación del mapa en la que en lugar de representar el tiempo que ha pasado en cada cuadrante, los valores representan simplemente si se ha visitado el cuadrante (valor 0.5) o no (valor 0), y si se encuentra en este momento en el cuadrante (valor 1).

El principal problema de esta percepción es que, en el caso de acciones basadas en cambio de velocidad (explicado más adelante) el agente no es consciente de su velocidad sólo por medio de la percepción. Esto se puede solucionar mediante el uso de "frame-stacking" o políticas recurrentes como LSTM que le puedan otorgar memoria.

Acciones

El diseño de las acciones es muy importante pues define completamente el MDP subyacente y puede dificultar el problema según se diseñe. Unas acciones demasiado complicadas o inconsistentes pueden provocar que el agente nunca encuentre una manera de utilizarlas de forma óptima.

En base a lo que permite AirSim, se proponen dos tipos de acciones. La primera consiste en ofrecer 7 acciones distintas, donde las 6 primeras representan una dirección a tomar (arriba, abajo, izquierda, derecha, delante y detrás) y la última representa "no hacer nada". Una vez seleccionada una acción, esta hará que aumente

la velocidad del dron en una pequeña cantidad en la dirección seleccionada. Esto permite al dron tomar altas o bajas velocidades en cualquier dirección según sea conveniente, pero con el inconveniente de que quizás añada demasiada complejidad al MDP. Otro inconveniente es que la percepción no ofrece información de la velocidad y requeriría al agente "adivinar" que velocidad lleva, o usar políticas recurrentes o utilizar "frame-stacking" para la entrada.

La segunda opción es la más simple y consiste en tomar una de las 6 direcciones, pero en este caso el dron siempre se desplaza una cantidad fija. El problema de esta opción es que el método simulado de AirSim para moverse una cantidad fija es inconsistente. La alternativa que se toma es usar un método no simulado que transporta el vehículo a una posición fija, asegurándose de que no traspasa superficies y todas las colisiones se tienen en cuenta. Este método es más parecido a un MDP clásico simple y gracias a su simplicidad tiene más posibilidades de converger a una política aceptable, pero estando siempre limitado a una velocidad fija.

Entorno	Acciones
Cambio de velocidad	$v_{t+1} = v_t + \{C_0, ..., C_5, 0\}$
Cambio de posición	$pos_{t+1} = pos_t + \{C_0, ..., C_5\}$

Cuadro 5.2: Entornos con diferentes sets de acciones. C_i es la constante definida por la acción i.

Reinicio e intercambio de entorno

En este problema se busca obtener un agente que sea capaz de explorar entornos que no haya visitado nunca. Para esto es necesario obtener cierto grado de generalización y esto se hace cambiando el entorno a uno nuevo cada reinicio. En la implementación, cada vez que se alcanza la condición de finalización, el dron se coloca en un entorno nuevo de un conjunto preseleccionado y se espera que .

La condición de finalización ocurre cuando se cumple una de las siguientes condiciones: (1) el vehículo ha colisionado con el entorno ó (2) el agente lleva 10 acciones sin realizar descubrimientos notables.

La segunda condición garantiza que el entorno se reinicie una vez se haya explorado todo pues una vez el mapa esté completo, no se descubrirán nuevos vóxels y no se generará recompensa positiva.

OpenAI Gym (framework)

El entorno completo se ha implementado basado en las especificaciones de Gym, más concretamente se usa la clase «Env» como superclase de la implementación del entorno.

OpenAI Gym es una popular librería para el lenguaje de programación Python que codifica un framework para la implementación de entornos para aprendizaje por refuerzo. Una de las principales características de la librería es que incluye varios entornos preparados para el entrenamiento de agentes y sirve como benchmark para el área del aprendizaje por refuerzo.

En este proyecto no se utilizan los entornos incluidos en Gym, si no que se aprovecha la superclase «Env» para la implementación de un entorno personalizado estandarizado para poder entrenar el piloto automático del dron. Esto es gracias a que esta clase codifica los métodos e información básicos para su uso general en aprendizaje por refuerzo.

Una de las principales ventajas de Gym es que al ser una librería de uso común en aprendizaje por refuerzo, otras librerías de algoritmos están preparadas para la interacción directa con sus entornos, ofreciendo así un estándar. Tal es el caso de la librería Standard Baselines que sólo acepta entornos de Gym [63].

5.4.2. Recompensa

El objetivo del aprendizaje por refuerzo es que la política maximice la recompensa acumulada del entorno. Para cumplir los objetivos del proyecto por lo tanto es necesario que se recompensen adecuadamente acciones que exploren y generen un mapa completo.

La recompensa creada se basa en algunos principios de "Reward Shaping" en [40], más específicamente en el principio de diseño de recompensa basada en "potencial", donde una buena función de recompensa premia acciones que acerquen al agente al objetivo de forma constante. Aquí se explica que la recompensa debe ser preferiblemente calculada tras cada acción en función de si le acerca al objetivo o no.

Un problema común a evitar en el diseño de recompensas es el "efecto cobra" que consiste en que las funciones de recompensa permitan al agente explotar la función de forma indeseada de tal manera que el agente se centre en generar recompensa sin realmente acercarse al objetivo.

La función de recompensa diseñada en este entorno premia el descubrimiento de nodos nuevos generados en el mapa OctoMap que representa directamente el entorno explorado. Solo se provee recompensa positiva cuando el agente se acerca al objetivo y garantiza que no se encuentren estrategias que abusen la recompensa de forma indeseada. Además, para incentivar mayor rapidez se crea mayor recompensa cuantos más vóxels sean descubiertos en un mismo paso hasta un valor de 4. El límite de recompensa se impone con intención de evitar que el agente priorice velocidad de generación del mapa, dejando de lado exploración más local. La función entonces para la recompensa positiva es:

$$r_t^+ = max(4, (voxels_t - voxels_{t-1}) \cdot C); C \in (0, 1) \tag{5.1}$$

A parte de esto, se penaliza al agente bruscamente cuando colisiona con el entorno (-15 de recompensa) y ligeramente cuando se sale del área de exploración (recompensa -1). La colisión implica el final del episodio y por lo tanto sólo hay un paso penalizado, es por esto que la recompensa debe ser muy brusca para permitir que aprenda más rápido como se explica en [40].

Situación	Valor de recompensa	Descripción
Nuevos vóxels	$r = r^+ \in [0, 4]$	Premia más descubrimientos.
Colisión con entorno	$r = -15$	Fin del episodio.
Fuera de límites	$r = -1$	Se sale de los límites de exploración.

Cuadro 5.3: Recompensas en escenario 1.

Como alternativa se propone con una función de recompensa basada en "potencial" [40] que consiste en que la recompensa se computa a partir del total de vóxels descubiertos en el OctoMap, en lugar de computarse a partir de los descubrimientos hechos en la última acción. Puesto que a priori se desconoce el total de vóxels que compondrían un entorno, no se puede colocar un techo fijo a la recompensa, pero se estima del orden de 10000 vóxels y con ello se estima la función de recompensa positiva $r^+ = n_{voxels} \cdot C$ donde se estima un valor C de tal manera que se espere que la recompensa no supere el valor 10. Por ejemplo, $C = 0{,}0005$.

La principal pega de esta segunda función es que no se controla la velocidad de exploración y el agente puede converger prematuramente a una política en la que prioriza la velocidad en lugar de la completitud o intente explotar la función y se quede estancado en una exploración muy local.

Situación	Valor de recompensa	Descripción
Vóxels totales	$r = r^+ \in [0, \infty)$	Techo desconocido.
Colisión con entorno	$r = -15$	Fin del episodio.
Fuera de límites	$r = -1$	Se sale de los límites de exploración.

Cuadro 5.4: Recompensas en escenario 2.

Cabe mencionar que tras testear OctoMap se ha encontrado que el framework no es completamente consistente con los vóxels encontrados y ocasionalmente pierde o gana vóxels de forma cíclica cuando el agente no se mueve. Es por esto que en la primera función de recompensa se le impone un valor mínimo de vóxels descubiertos para evitar que el agente abuse esta inconsistencia y permitir que la condición de final basado en la recompensa se pueda activar.

5.4.3. Algoritmo y arquitectura

El aprendizaje se realiza con el algoritmo PPO. Como se explica en Sección 4.4, es un algoritmo de clase "actor-critic" (requiere arquitectura dividida) y ha demostrado ser uno de los mejores algoritmos de aprendizaje por refuerzo que gracias a su uso de "clipping", destaca por ser especialmente consistente en diferentes problemas. Otra de las ventajas de PPO es su uso reducido de memoria al no requerir un "replay-buffer" muy grande.

Otro de los principales motivos por los que se selecciona PPO como optimizador, es que destaca por ser más fácil de afinar hiperparámetros. Esto es importante porque el entrenamiento en este entorno es muy lento (pudiendo llegar a varios días por agente) y por ello se necesita minimizar el número de versiones a entrenar dejando poco margen para afinar hiperparámetros.

Arquitecturas

Como todos los algoritmos de Deep Reinforcement Learning, se requiere una arquitectura a optimizar. En el caso de algoritmos tipo actor-critic la arquitectura requiere una output para la función de valor y otra separada para las acciones (política), y esto normalmente se realiza con una arquitectura dividida o directamente dos redes distintas.

Otro factor a tener en cuenta es que la input se compone de una imagen y vector separados lo cuál se resuelve con una arquitectura "multi-input". Las arquitecturas que se proponen están basada en la arquitectura usada en [64] donde una primera input de la imagen se transforma a un vector mediante capas de convolución y pooling, la segunda input se procesa mediante una sola capa y luego ambos vectores resultantes se concatenan y procesan como un sólo vector de características latentes. Estas primeras capas se comparten tanto para la función de valor como para la política como un "extractor de características" (feature extractor) compartido. Después el vector de características se procesa por capas feed-forward separadas para producir la output de la función de valor y de la política.

Se proponen dos arquitecturas distintas: La primera es una de las arquitecturas implementadas de serie en la librería Standard Baselines 3, más concretamente la arquitectura actor-critic multi-input [65] diseñada específicamente para observaciones multi-input que está formada parcialmente por la arquitectura "Nature Atari CNN" para la extracción de características latentes de las imágenes y luego concatenarlas con la input vector y ser posteriormente procesada. Se decide usar esta arquitectura porque está testeada en diversos problemas y tiene garantizado cierto nivel de rendimiento.

La segunda arquitectura está diseñada en la librería PyTorch de cero e incorpora nuevas capas y una capa recurrente LSTM (Long Short Term Memory) que le otorga

memoria al agente. Esta también usa un extractor de características compartido para las inputs que luego se divide en varias capas. La capa LSTM es la capa final del extractor de características. Se espera que al añadir capas adicionales la política pueda converger a estrategias más complejas.

En ambas arquitecturas se usa un extractor de características compartido junto a un diseño minimalista para reducir el número de parámetros y así minimizar el tiempo de entrenamiento. Esto es importante debido a que usando AirSim un agente puede tardar días en entrenarse completamente.

PyTorch y Standard Baselines 3 (frameworks)

En este trabajo se usan las librerías PyTorch y Standard Baselines 3 (sb3). PyTorch es, compitiendo con TensorFlow, una de las librerías líder para la implementación de todo tipo de arquitecturas y algoritmos de Deep Learning.

PyTorch incluye implementaciones de bajo nivel de la mayoría de algoritmos estado del arte en todo tipo de áreas en Deep Learning (aprendizaje supervisado, aprendizaje por refuerzo, CNNs, redes neuronales de grafos, etc.), y es la librería que se usa para implementar la segunda arquitectura propuesta para la navegación autónoma del dron en este trabajo.

Standard Baselines 3 es un framework de alto nivel para aprendizaje por refuerzo que, usando PyTorch y Gym, implementa soluciones completas de aprendizaje por refuerzo. Aunque es muy fácil de usar, también sacrifica flexibilidad y, por ejemplo, actualmente no permite la implementación de arquitecturas recurrentes (aunque se planea para el futuro).

6. RESULTADOS Y DISCUSIÓN

En este capítulo se presentan y discuten los resultados generales del trabajo que incluye la implementación de una herramienta para mapeo tridimensional con vehículos, un entorno estandarizado para entrenamiento de aprendizaje por refuerzo en simulador y los resultados obtenidos con los agentes creados.

6.1. Herramienta de mapeo

La herramienta diseñada (véase Sección 5.3) está pensada para su uso en el entorno de aprendizaje por refuerzo y es por esto que utiliza OctoMap con la intención de permitir la generación de un mapa temporal ligero que permita operaciones rápidas sobre lo explorado. Pero a parte de esto la herramienta es capaz fácilmente de extraer representaciones con mayor detalle.

En general esta herramienta recibe los datos de los haces emitidos por LIDAR (aunque también pueden ser haces a partir de otros sensores como radar), y procesa y almacena los datos pertinentes para representar el mapa, permitiendo finalmente visualizarlo tanto en su versión simplificada de OctoMap como en su versión completa de nube de puntos.

Existen dos maneras de extraer representaciones de alto detalle del entorno mapeado. La primera consiste en almacenar en bruto toda la nube de puntos y visualizarla con la herramienta incluida. Posteriormente estos puntos pueden ser procesados para eliminar exceso indeseado o cambiar su formato de representación.

La segunda manera consiste en aumentar la resolución del OctoMap al nivel deseado. Esta solución permite crear un mapa de vóxels detallado, pero a coste de aumentar rápidamente la complejidad temporal de las operaciones usadas posteriormente por el entorno. Una alternativa para usar esta solución con un agente autónomo es crear un OctoMap adicional de alta resolución para almacenar la nube de puntos de forma paralela.

Técnicamente esta herramienta podría ser usada con algunas modificaciones en drones reales con pilotaje manual, pero requeriría de alguna manera de enviar los datos de visualización al usuario para que este pueda ver el progreso del mapeo y tomar las acciones adecuadas para obtener un mapa completo. Una alternativa sería ejecutar el algoritmo remotamente, pero requeriría transmitir todos los datos LIDAR de forma inalámbrica.

6.2. Entorno de RL estandarizado

El entorno de aprendizaje por refuerzo implementado (véase Subsección 5.4.1) ofrece una funcionalidad completa de tal manera que siga la interfaz estándar de la librería gym y de esta manera ofrece un entorno compatible con la mayoría de implementaciones de algoritmos de aprendizaje por refuerzo.

El entorno ha sido testeado para minimizar inconsistencias o errores y tras varios días de entrenamiento continuo no ha fallado. Una inconsistencia existente proviene en ciertas situaciones de la actualización de vóxels que aparecen y desaparecen en el OctoMap, que puede causar un pequeño margen de error en la recompensa.

A causa de los resultados subóptimos obtenidos para los agentes autónomos, es posible que la percepción o la recompensa (sus dos variaciones) no sean lo suficientemente buenas para resolver el problema específico. Este problema se discute con más detalle en la siguiente sección y en el último capítulo.

6.3. Resultados agentes autónomos

Para la obtención del agente autónomo de mapeo se han entrenado varios agentes durante varios días bajo las configuraciones especificadas en Subsección 5.4.3, con el objetivo de obtener un agente capaz de explorar y mapear los entornos con la mayor completitud posible. Gran parte de los entrenamientos se han realizado para afinar parámetros en las diferentes configuraciones propuestas, pero en general, la mayoría de resultados han sido similares.

Las configuraciones de agentes y entorno entrenados varían según set de acciones, según la función de recompensa y según la arquitectura usada.

	Set acciones 1	Set acciones 2
Recompensa 1	A y B	A y B
Recompensa 2		A

Cuadro 6.1: Configuraciones de entorno probadas para las arquitecturas A (feed-forward) y B (recurrente). Set acciones 1 basado en velocidad y 2 en posición. Recompensa 1 basada en descubrimiento, 2 en tamaño total de exploración.

En Figura 6.1 se muestran las gráficas de recompensa para las diversas configuraciones para la primera función de recompensa diseñada (véase Subsección 5.4.2). Sólo se muestran las mejores iteraciones para cada arquitectura probada. La causa de la volatilidad entre episodios se debe a que el entorno cambia cada episodio provocando que el agente pueda rendir de manera muy distinta a la anterior y posiblemente afectando al aprendizaje.

En el mejor caso la recompensa total acumulada por episodio alcanza alrededor

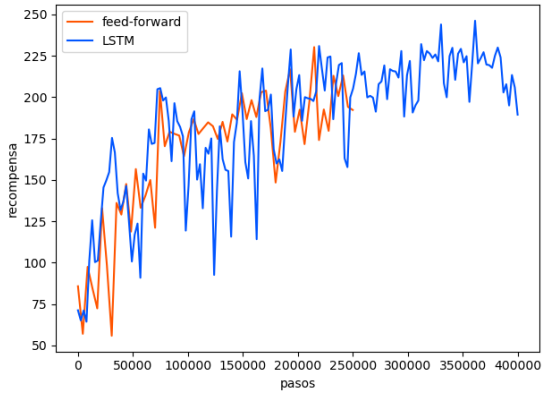


Figura 6.1: Recompensas obtenidas durante el entrenamiento por las mejores iteraciones de la arquitectura feed-forward (naranja) y LSTM (azul) bajo la primera configuración de recompensa. Se asume convergencia temprana pues iteraciones anteriores también convergen en esos valores.

de 250 de recompensa consistentemente, pero desgraciadamente no es un buen resultado. En Figura 6.2 se muestra un ejemplo de un entorno mapeado por el mejor agente (los haces LIDAR tienen un alcance bajo). Como se puede ver el mapa tiene secciones inexploradas y las exploradas tienen algunas áreas incompletas. Aunque las zonas exploradas tienen cierto grado de completitud, estos agentes se consideran un fracaso.

Se valora la posibilidad de que con más tiempo de entrenamiento alguna configuración pudiera escapar de la estrategia local, pero se descarta debido a los limitados recursos y a partir de haber probado entrenamientos más largos en primeras iteraciones sin salir de la misma estrategia.

Para la segunda configuración de recompensa sólo se entrena un agente con la arquitectura A (feed-forward). Esto es debido a que lleva demasiado tiempo, especialmente en este entorno que requiere más del doble de pasos para converger. Los resultados son ligeramente mejores ofreciendo una exploración local más detallada pero sigue sufriendo el mismo problema de explorar una sola dirección.

Analizando los resultados se cree que es posible en esta segunda configuración obtener mejores agentes con arquitecturas más complejas, pero en este caso no se dispone de más tiempo para entrenarlas.

6.3.1. Problema de convergencia prematura

En todos los casos la causa de los malos resultado es clara: los agentes entrenados convergen prematuramente a estrategias simples que dejan sin explorar porciones muy grandes del mapa. La estrategia más común a la que convergen los agentes es tomar una linea más o menos recta hasta el borde del mapa que les garantiza una

Figura 6.2: Mapa de bajo detalle generado por el mejor agente. Tiende a explorar en una sola dirección.

recompensa alta y consistente hasta que alcancen el límite del mapa. Tras muchos pasos de entrenamiento, los agentes sólo parecen aprender a optimizar ligeramente esa estrategia.

La causa principal de esta convergencia se puede deber a la aparición de grandes "depresiones" en el espacio de la función de recompensa del POMDP, provocando que un número limitado de acciones aleatorias difícilmente encuentren subidas notables en la recompensa y sea por ende, complicado encontrar la salida de esa depresión. En otras palabras, para que el agente descubra que cambiar de trayectoria en el borde del mapa es mejor que seguir recto o quedarse quieto, sería necesario que tomase varias acciones seguidas en una dirección perpendicular a la previa para que los haces LIDAR puedan alcanzar suficiente terreno inexplorado y así obtener cambios notables en la función de recompensa. Todo esto es empeorado por el hecho de que tras cada episodio el entorno cambia.

En conclusión, la función de recompensa posiblemente no tenga un espacio lo suficientemente "suave" para que PPO pueda encontrar trayectorias mejores. Esto quizás se pueda solucionar retocando la función de recompensa o encontrando la arquitectura e hiperparámetros adecuados, pero para los recursos disponibles en este trabajo (y con mi limitada experiencia) no ha sido posible.

Otra idea habría sido cambiar completamente la estrategia del entorno y usar una recompensa distinta (por ejemplo, puramente basada en exploración ignorando LIDAR), o usar una estrategia jerárquica.

7. MARCO REGULADOR Y ENTORNO SOCIO-ECONÓMICO

7.1. Marco regulador

7.1.1. Legislación

El software usado en este trabajo (OctoMap, AirSim, etc.) son proyectos de código abierto que se rigen sobre licencias específicas. En todos los casos se respetan en este trabajo estas licencias.

Para el uso de drones, las regulaciones europeas 2019/947 y 2019/945 establecen reglas a seguir para la operación de drones civiles. Estas regulaciones existen con la finalidad de garantizar la operación segura de drones y tienen en cuenta el peso y tamaño de los drones.

Además, pueden existir leyes particulares para cada país existen leyes regulando el uso de drones. En general, es posible que para la aplicación real de mapeo de áreas con drones se requiera pedir permiso o que no sea una zona restringida.

El mapeo tridimensional de superficies en áreas indeseadas puede comprometer derechos de privacidad o propiedad intelectual de individuos u organizaciones e infringir leyes como la ley de protección de datos europea.

En general, no hay legislación especifica sobre el uso de LIDAR en Europa.

7.1.2. Estándares técnicos

Los estándares y regularización de la inteligencia artificial se encuentran actualmente en fases tempranas. IEEE tiene la "Global Initiative on Ethics of Autonomous and Intelligent Systems" (Iniciativa global en ética de sistemas autónomos e inteligentes) que cubre ciertas preocupaciones sociales en la inteligencia artificial como parcialidad en IAs o temas de privacidad. También existe el ISO/JTC 1 SC 42, que cubre ciertas preocupaciones de seguridad.

Los sensores LIDAR utilizan el formato estándar LAS para el almacenamiento de los datos.

Para la implementación del proyecto en el lenguaje de programación Python, se ha seguido la guía estándar PEP 8 "Style Guide for Python Code". Esta guía determina un estilo consistente de programación en el lenguaje y facilita la comprensión del código.

7.2. Entorno socio-económico

7.2.1. Presupuesto del trabajo

Se indican los costes directos e indirectos ocurridos a causa de este trabajo. Puesto que muchos precios no están fijos (como el coste de tiempo de trabajo) algunos de estos costes indicados son meramente simbólicos.

Costes directos

Se asume un precio de la electricidad de 0,2eur/kWh.

Puesto que gran parte de este trabajo se ha realizado a tiempo completo, el coste de trabajo se tiene en cuenta con motivo de los costes de vida y de incapacidad de trabajar mientras se realiza este proyecto. Para esto se asume un coste simbólico de 10eur/día y se indica como "coste labor".

Todos los valores indicados son estimados y no precisos.

Computación entrenamiento agentes.	288h@500W	28,8eur
Computadora otros usos.	300h@300W	9eur
Costes labor.	100 días	1000eur

Cuadro 7.1: Costes directos de la realización del trabajo.

A un total de 1037,8 euros como costes directos.

Costes indirectos

Se añade un coste indirecto del desgaste de la computadora usada con un porcentaje arbitrario del 5 %. A un coste aproximado de 800eur, el coste por desgaste es de 40eur.

Para desgaste de otros periféricos y mobiliario se añade un coste arbitrario de 20eur, saliendo un total de 60eur de costes indirectos.

7.2.2. Impacto socio-económico

Este trabajo explora bases para la creación de sistemas autónomos de mapeo tridimensional.

Como se ha discutido en secciones anteriores, existe un interés comercial en el mapeo tridimensional de entornos, particularmente haciendo uso de UAVs. Un sistema autónomo de mapeo supondría un recorte directo a los costes humanos para la producción de los mapas al no requerir piloto.

Un producto útil inmediato de este proyecto es la herramienta de mapeo tridimensional que puede ser adaptada con relativa facilidad para su uso en el mundo real bajo pilotaje humano. Esto podría servir de alternativa a otras herramientas de mapeo con licencias propietarias.

La automatización del proceso de mapeo si posible, podría ofrecer claras ventajas a las organizaciones y empresas. Aunque este trabajo no soluciona directamente el problema, ofrece un acercamiento a una primera solución.

Además, la simplicidad de la solución explorada sugiere mayores posibilidades de accesibilidad a herramientas para el mapeo tridimensional de entornos para organizaciones más pequeñas o incluso individuos.

8. CONCLUSIONES Y TRABAJO FUTURO

El mapeo tridimensional de terrenos no es un área de investigación muy activa pero de interés para algunas tareas específicas e incluso de cierto interés comercial. Aunque existen soluciones capaces de mapear terrenos tridimensionales con detalle usando UAVs, estas suelen ser propietarias y la navegación de los vehículos no son autónomos requiriendo un operador con cierto nivel de habilidad. Alternativamente existen soluciones autónomas que mapean su entorno, muy notablemente en problemas "SLAM", pero el mapeo no suele ser su finalidad si no un medio y con ello generan soluciones de limitado detalle y muchas veces con imprecisiones.

Este trabajo, bajo algunas asunciones de cara al futuro, investiga y junta algunas ideas y trabajos previos para la creación de herramientas que permitan mapear terrenos en tres dimensiones y de forma autónoma con requisitos computacionales reducidos.

La investigación realizada en la primera parte del trabajo trata sobre la tecnología LIDAR, los UAVs (específicamente drones pequeños) y problemas de navegación autónoma. Estas tecnologías se encuentran actualmente en un crecimiento muy rápido y sugiere la posibilidad de que en el futuro sea posible mapear terrenos con alto nivel de detalle en drones pequeños usando sensores LIDAR. Es a partir de estas conclusiones que se establecen los objetivos de este trabajo.

Se crea, en simulador, un sistema completo que, con un algoritmo de aprendizaje por refuerzo, genera un mapa tridimensional de forma completamente autónoma, demostrando la posibilidad de hacer esto con mayor o menor calidad.

Gran parte del trabajo consisten en investigar e implementar un agente de aprendizaje por refuerzo. Aunque el entorno creado funciona satisfactoriamente y con buen rendimiento, el agente autónomo no alcanza el nivel de ser de utilidad, pero después de estudiar los resultados, se cree que es posible obtener mejores agentes realizando los cambios adecuados.

También es posible que directamente no sea viable resolver este problema mediante aprendizaje por refuerzo. Aunque esta rama del aprendizaje automático ha conseguido hazañas monumentales logrando hasta batir a jugadores profesionales en complejos videojuegos como Dota 2, esto se ha conseguido durante meses de computación en superordenadores para arquitecturas mucho más complejas. Es sabido que la desproporcionada cantidad de datos que el aprendizaje por refuerzo necesita es uno de sus mayores problemas y los fallidos resultados de este trabajo pueden ser una causa de ello.

Trabajo futuro

Este trabajo sirve como prueba de concepto en el mejor de los casos o como conjunto de ideas y frameworks útiles en el peor. Aunque no es capaz de ofrecer un sistema directamente útil en el mundo real, ofrece varios recursos que podrían ser reutilizados para trabajos similares.

Este trabajo investiga y reúne frameworks existentes para resolver los problemas de diseño que aparecen y prueba ideas para combinarlos todos. Todos estos recursos pueden ser reutilizados para la creación de herramientas relacionadas o de una solución mejor a la presentada.

También se realiza una investigación resumiendo cierta información relevante con citas sobre todas las tecnologías involucradas que podría ser reusada para trabajos futuros. Aunque se admite que es muy posible que la investigación pueda ser de baja calidad al contener potencialmente fallos o demasiadas asunciones.

Una mejora directa a las herramientas diseñadas sería combinar otras técnicas de fotogrametría para añadir color a los mapas generados, que ya se ha demostrado posible varias veces en el pasado.

BIBLIOGRAFÍA

[1] G. Jozkowa, C. Totha y D. Grejner-Brzezinska, "UAS topographic mapping with Velodyne LiDAR sensor", *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 3, n.º 1, Julio 2016, pp.201-208. [En linea]. https://doi.org/10.5194/isprs-annals-III-1-201-2016 (acceso: 4 de septiembre de 2021).

[2] A. Walford. "What is Photogrammetry?". photogrammetry.com. http://www.photogrammetry.com/ (acceso: 4 de septiembre de 2021).

[3] Flyability. Elios 2: página de producto. flyability.com. https://www.flyability.com/es (acceso: 4 de septiembre de 2021).

[4] Pix4D. Página principal. pix4d.com. https://www.pix4d.com/es (acceso: 4 de septiembre de 2021).

[5] FlyGuys. Drone Services for Surveying & Mapping. flyguys.com. https://flyguys.com/drone-services/drone-mapping-services/ (acceso: 4 de septiembre de 2021).

[6] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I", *IEEE Robotics & Automation Magazine*, vol. 13, n.º 2, pp.99-110, Junio 2006, [En linea] Disponible en: https://doi.org/10.1109/MRA.2006.1638022 (acceso: 4 de septiembre de 2021).

[7] L. von Stumberg, V. Usenko, J. Engel, J. Stückler y D. Cremers, "From monocular SLAM to autonomous drone exploration", *2017 European Conference on Mobile Robots (ECMR)*, pp.1-8, 2017. [En linea] Disponible en: https://doi.org/10.1109/ECMR.2017.8098709 (acceso: 4 de septiembre de 2021).

[8] NOAA. "What is lidar?". National Ocean Service website. https://oceanservice.noaa.gov/facts/lidar.html (acceso: 4 de septiembre de 2021).

[9] L. A. Wasser. "The Basics of LiDAR - Light Detection and Ranging - Remote Sensing". National Science Foundation's National Ecological Observatory Network. https://www.neonscience.org/resources/learning-hub/tutorials/lidar-basics (acceso: 4 de septiembre de 2021).

[10] A. Cerrillo. "Formato LAS, el estándar de datos LiDAR". LiDAR.com.es. http://lidar.com.es/2010/11/18/formato-las-el-estandar-de-datos-lidar/ (acceso: 4 de septiembre de 2021).

[11] J. Hecht, "Lidar for self-driving cars", *Optics & Photonics News*, vol. 29, nº. 1, pp.26-33, enero 2018. [En línea] Disponible en: https://www.osapublishing.org/opn/issue.cfm?volume=29&issue=1 (acceso: 4 de septiembre de 2021).

[12] L. Solomon. "LIDAR: The Speed Enforcement Weapon of Choice". OFFICER.com. https://www.officer.com/on-the-street/article/10250592/lidar-the-speed-enforcement-weapon-of-choice (acceso: 4 de septiembre de 2021).

[13] S.Y. BenZvil et al., "The Lidar system of the Pierre Auger Observatory", *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 574, n.º 1, pp.171-184, 21 April 2007. [En linea] Disponible en: https://doi.org/10.1016/j.nima.2007.01.094 (acceso: 4 de septiembre de 2021).

[14] M.Mutlu, S.C. Popescu, C. Stripling y T. Spencer, "Mapping surface fuel models using lidar and multispectral data fusion for fire behaviour", *Remote Sensing of Environment*, vol. 112, n.º 1, pp.274-285, enero 2008. [En linea] Disponible en: https://doi.org/10.1016/j.rse.2007.05.005 (acceso: 4 de septiembre de 2021).

[15] K. Koenig, B. Höfle, M. Hämmerle, T. Jarmer, B. Siegmann y H. Lilienthal, "Comparative classification analysis of post-harvest growth detection from terrestrial LiDAR point clouds in precision agriculture", *ISPRS Journal of Photogrammetry and Remote Sensing* vol. 104, pp.112-125, Junio 2015. [En linea] Disponible en: https://doi.org/10.1016/j.isprsjprs.2015.03.003 (acceso: 4 de septiembre de 2021).

[16] T. Wilkerson, I. Andrus, J. Sanders, G. Schwemmer, D. Miller, D. Guerra y D.OC. Starr, "Horizontal Wind Measurements using the HARLIE Holographic Lidar", presentada en el 46th Annual SPIE Meeting, San Diego, California, enero 2001. [En linea] Disponible en: https://ntrs.nasa.gov/citations/20010079653 (acceso: 4 de septiembre de 2021).

[17] Lista de especificaciones de sensores LiDAR. Hexagon, Autonomy and Positioning Division. https://autonomoustuff.com/lidar-chart (acceso: 4 de septiembre de 2021).

[18] T.H. Maiman, "Stimulated optical radiation in ruby", *Nature*, vol. 187, pp.493–494, agosto 1960. [En linea] Disponible en: https://doi.org/10.1038/187493a0 (acceso: 4 de septiembre de 2021).

[19] J.L. Bromberg, The Laser in America, 1950–1970. Cambridge, MA: The MIT Press, 1991, pp.74–77.

[20] "New Radar System", *Odessa American*, 28 Feb 1961.

[21] G.G. Goyer y R. Watson, *Bulletin of the American Meteorological Society*, vol. 44, n.º 9, pp.564–570, Septiembre 1963. [En linea] Disponible en: https://doi.org/10.1175/1520-0477-44.9.564 (acceso: 4 de septiembre de 2021).

[22] J.B. Abshire, "NASA's Space Lidar Measurements of Earth and Planetary Surfaces", NASA. [En linea] Disponible en: https://ntrs.nasa.gov/citations/20100031189 (acceso: 4 de septiembre de 2021).

[23] BCC Research Editorial. "Brief History of LiDAR, Its Evolution and Market Definition". bcc Research. https://blog.bccresearch.com/brief-history-of-lidar-evolution-and-market-definition (acceso: 4 de septiembre de 2021).

[24] A. J. Davison, I. D. Reid, N. D. Molton y O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052-1067, Junio 2007, [En linea] Disponible en: https://doi.org/10.1109/TPAMI.2007.1049 (acceso: 4 de septiembre de 2021).

[25] Velodyne. Velodyne HDL-32E: página de producto. Velodyne Lidar website. https://velodynelidar.com/products/hdl-32e/ (acceso: 4 de septiembre de 2021).

[26] DJI. DJI Zenmuse L1: página de producto. DJI website. https://www.dji.com/es/zenmuse-l1 (acceso: 4 de septiembre de 2021).

[27] Microsoft. AirSim Drone Racing Lab: Repositorio de código fuente. GitHub. https://github.com/microsoft/AirSim-Drone-Racing-Lab (acceso: 4 de septiembre de 2021).

[28] C. Luo, S.X. Yang, M. Krishnan, M. Paulik y Y. Chen, "A hybrid system for multi-goal navigation and map building of an autonomous vehicle in unknown environments" *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2013, pp. 1228-1233. [En linea] Disponible en: https://doi.org/10.1109/ROBIO.2013.6739632 (acceso: 4 de septiembre de 2021).

[29] S.U. Kamat y K. Rasane, "A Survey on Autonomous Navigation Techniques", *2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC)*, 2018, pp. 1-6. [En linea] Disponible en: https://doi.org/10.1109/ICAECC.2018.8479446 (acceso: 4 de septiembre de 2021).

[30] B. Li y Y. Wu, "Path Planning for UAV Ground Target Tracking via Deep Reinforcement Learning", *IEEE Access*, vol. 8, pp. 29064-29074, 2020, [En linea] Disponible en: https://doi.org/10.1109/ACCESS.2020.2971780 (acceso: 4 de septiembre de 2021).

[31] A.T. Azar et al., "Drone Deep Reinforcement Learning: A Review", *Electronics*, vol. 10, n.º 9, p. 999, Abril 2021. [En linea] Disponible en: https://doi.org/10.3390/electronics10090999 (acceso: 4 de septiembre de 2021).

[32] B.G. Maciel-Pearson, L. Marchegiani, S. Akçay, A. Atapour-Abarghouei, J. Garforth y T.P. Breckon, "Online Deep Reinforcement Learning for Autonomous UAV Navigation and Exploration of Outdoor Environments", diciembre 2019. [En linea] Disponible en: https://arxiv.org/pdf/1912.05684.pdf (acceso: 4 de septiembre de 2021).

[33] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss y W. Burgard, "OctoMap: an efficient probabilistic 3d mapping framework based on octrees", *Autonomous Robots*, vol. 34, nº. 3, pp.189–206, abril 2013. [En linea] Disponible en: https://doi.org/10.1007/s10514-012-9321-0 (acceso: 4 de septiembre de 2021).

[34] R.S. Sutton y A.G. Barto, Reinforcement Learning: An Introduction, 2ª ed. Cambridge, Massachusetts, London: The MIT Press, 2014. [En línea]. Disponible en: https://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf (acceso: 4 de septiembre de 2021).

[35] R.S. Sutton y A.G. Barto, "Psychology" en Reinforcement Learning: An Introduction, 2ª ed. Cambridge, Massachusetts, London: The MIT Press, 2014, pp.341-370.

[36] A. Dickinson, "Actions and Habits: The Development of Behavioural Autonomy", *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, vol. 308, n.º 1135, pp. 67–78, 1985. [En línea]. Disponible en: www.jstor.org/stable/2396284 (acceso: 4 de septiembre de 2021).

[37] R.S. Sutton y A.G. Barto, "Finite Markov Decision Processes" en Reinforcement Learning: An Introduction, 2ª ed. Cambridge, Massachusetts, London: The MIT Press, 2014, 47-72.

[38] M.L. Littman, "Markov Decision Processes", *International Encyclopedia of the Social & Behavioral Sciences*, pp.9240-9242, 2001. [En línea]. Disponible en: https://doi.org/10.1016/B0-08-043076-7/00614-8 (acceso: 4 de septiembre de 2021).

[39] J.D. Williams y S. Young, "Partially observable Markov decision processes for spoken dialog systems", *Computer Speech & Language*, vol. 21, n.º 2, 2007, pp.393-422, [En línea]. Disponible en: https://doi.org/10.1016/j.csl.2006.06.008 (acceso: 4 de septiembre de 2021).

[40] P. Mannion, S. Devlin, K. Mason, J. Duggan y E. Howley, "Policy invariance under reward transformations for multi-objective reinforcement learning",

Neurocomputing, vol. 263, pp.60-73, noviembre 2017. [En línea]. Disponible en: https://doi.org/10.1016/j.neucom.2017.05.090 (acceso: 4 de septiembre de 2021).

[41] R.S. Sutton y A.G. Barto, "Temporal-Diference Learning" en Reinforcement Learning: An Introduction, 2ª ed. Cambridge, Massachusetts, London: The MIT Press, 2014, pp.119-140.

[42] J. Gläscher, N. Daw, P. Dayan, J.P. O'Doherty, "States versus Rewards: Dissociable Neural Prediction Error Signals Underlying Model-Based and Model-Free Reinforcement Learning", *Neuron*, vol. 66, n.º 4, pp.585-595, mayo 2010, [En línea]. Disponible en: https://doi.org/10.1016/j.neuron.2010.04.016 (acceso: 4 de septiembre de 2021).

[43] D.P. Kroese, T. Brereton, T. Taimre y Z.I. Botev, "Why the Monte Carlo method is so important today", *WIREs Comput Stat*, vol. 6, n.º 6, pp.386–392. [En línea]. Disponible en: https://doi.org/10.1002/wics.1314 (acceso: 4 de septiembre de 2021).

[44] B. Roy. "Monte Carlo Learning". Towards Data Science. https://towardsdatascience.com/monte-carlo-learning-b83f75233f92 (acceso: 4 de septiembre de 2021).

[45] D.P. Bertsekas y J.N. Tsitsiklis, Neuro-Dynamic Programming. Belmont, MA: Athena Scientific, 1996. [En línea]. Disponible en: https://doi.org/10.1007/978-0-387-74759-0_440 (acceso: 4 de septiembre de 2021).

[46] R.S. Sutton, D. McAllester, S. Singh y Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation", *Advances in Neural Information Processing Systems 12*, pp. 1057–1063, febrero 2000. [En línea]. Disponible en: https://dl.acm.org/doi/10.5555/3009657.3009806 (acceso: 4 de septiembre de 2021).

[47] R.S. Sutton y A.G. Barto, "Policy Gradient Methods" en Reinforcement Learning: An Introduction, 2ª ed. Cambridge, Massachusetts, London: The MIT Press, 2014, 321-338.

[48] IBM Cloud Education. Deep Learning. IBM website. https://www.ibm.com/cloud/learn/deep-learning (acceso: 4 de septiembre de 2021).

[49] D. Silver. Deep Reinforcement Learning. DeepMind. https://deepmind.com/blog/article/deep-reinforcement-learning (acceso: 4 de septiembre de 2021).

[50] A. El-Fakdi, M. Carreras and P. Ridao, "Towards Direct Policy Search Reinforcement Learning for Robot Control", *2006 IEEE/RSJ International Conference*

on Intelligent Robots and Systems, pp. 3178-3183, 2006. [En línea]. Disponible en: https://doi.org/10.1109/IROS.2006.282342 (acceso: 4 de septiembre de 2021).

[51] R.J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning", *Machine Learning*, vol. 8, pp.229–256, 1992. [En línea]. Disponible en: https://doi.org/10.1007/BF00992696 (acceso: 4 de septiembre de 2021).

[52] R. Antonin y contribuidores, "rl-zoo". Repositorio de Github, 2018. Disponible en: https://github.com/araffin/rl-baselines-zoo

[53] V. Mnih et al., "Playing Atari with Deep Reinforcement Learning", *NIPS Deep Learning Workshop 2013*, diciembre 2013. [En línea]. Disponible en: https://arxiv.org/abs/1312.5602 (acceso: 4 de septiembre de 2021).

[54] H. Hasselt, A. Guez y D. Silver, *AAAI 2016*, septiembre 2015. [En línea]. Disponible en: https://arxiv.org/abs/1509.06461 (acceso: 4 de septiembre de 2021).

[55] J. Schulman, F. Wolski, P. Dhariwal, A. Radford y O. Klimov, Proximal Policy Optimization Algorithms, arXiv. [En línea]. Disponible en: https://arxiv.org/abs/1707.06347 (acceso: 4 de septiembre de 2021).

[56] Volodymyr Mnih et al., "Asynchronous Methods for Deep Reinforcement Learning", presentada en 33rd International Conference on Machine Learning, New York, Estados Unidos, 2016. [En línea]. Disponible en: https://arxiv.org/abs/1602.01783 (acceso: 4 de septiembre de 2021).

[57] J. Schulman, S. Levine, P. Moritz, M.I. Jordan y P. Abbeel, "Trust Region Policy Optimization", presentada en 31st International Conference on Machine Learning, Lille, Francia, 2015. [En línea]. Disponible en: https://arxiv.org/abs/1502.05477 (acceso: 4 de septiembre de 2021).

[58] Z. Wang et al., "Sample Efficient Actor-Critic with Experience Replay", presentada en 2017 International Conference on Learning Representations, 2017. [En línea]. Disponible en: https://arxiv.org/abs/1611.01224 (acceso: 4 de septiembre de 2021).

[59] Microsoft. Documentación de AirSim. GitHub. https://microsoft.github.io/AirSim/ (acceso: 4 de septiembre de 2021).

[60] Microsoft. Documentación de AirSim, "Reinforcement Learning in AirSim". GitHub. https://microsoft.github.io/AirSim/reinforcement_learning/ (acceso: 4 de septiembre de 2021).

[61] Msgpack. Documentación de msgpack-rpc. msgpack.org. http://msgpack.org/rpc/rdoc/current/MessagePack/RPC.html (acceso: 4 de septiembre de 2021).

[62] K.M. Wurm et al., "OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems", presentado en 2010 IEEE International Conference on Robotics and Automation (ICRA), 2010. [En línea]. http://ais.informatik.uni-freiburg.de/publications/papers/wurm10octomap.pdf (acceso: 4 de septiembre de 2021).

[63] Documentación de Stable Baselines: "Using Custom Environments". Readthedocs. https://stable-baselines.readthedocs.io/en/master/guide/custom_env.html (acceso: 4 de septiembre de 2021).

[64] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz y D. Quillen, "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection", *The International Journal of Robotics Research*, vol 37, n.º 4-5, 2018. [En línea]. https://doi.org/10.1177/0278364917710318 (acceso: 4 de septiembre de 2021).

[65] Documentación de Stable Baselines: "Custom Policy Network". Readthedocs. https://stable-baselines3.readthedocs.io/en/master/guide/custom_policy.html (acceso: 4 de septiembre de 2021).

[66] Imagen por "Megajuice" - Trabajo Propio, CC0, https://commons.wikimedia.org/w/index.php?curid=57895741

[67] Imagen por Waldoalvarez - Trabajo Propio, CC BY-SA 4.0, https://upload.wikimedia.org/wikipedia/commons/a/ad/Markov_Decision_Process.svg

ANEXO A. ENGLISH SUMMARY

Introduction

Thanks to the latest advancements in technologies like LIDAR and UAVs, a new trend in mapping using drones and LIDAR has been appearing. This mapping is usually done from an aerial view for the creation of vertical 2D maps or distributions of some kind, but there is also a niche for three-dimensional mapping of more or less quality for projects such as Google Maps or Microsoft Flight Simulator.

Currently most three-dimensional mapping operations use techniques of photogrammetry with photographic cameras to infer volume from still photos at different angles and positions. Even it being a cheaper technique and allowing for the capture of color, it can lack precision and perform badly under certain conditions such as low lighting.

LIDAR technology is improving at high rate and is known for its reliability and precision when properly used and can be installed in UAVs. It has been proved to work in topographical mapping applications and has also been used alongside photographic imagery to improve precision in certain applications. LIDAR also has the advantage of not requiring good lighting or weather conditions.

Nowadays even small drones are capable of operating LIDAR sensors, with the main problem being lack of precision from the positioning sensors (IMUs, GPS) that have been improving lately as well. In the near future it might be possible to assemble a LIDAR drone with high precision and reliability.

Autonomous vehicle navigation is also an actively researched problem with existing solutions for problems such as obstacle avoidance, path planning or unknown environment exploration.

In this work the problem of high detail three-dimensional mapping is explored using small drones and combined with an attempt for autonomous navigation for exploration and mapping. A solution is implemented and demonstrated in simulator.

State of the art: 3D mapping, LIDAR and autonomous UAV navigation

First, the current state of terrain mapping is investigated. There is a commercial interest for 3D mapping for small (interiors, plots of land) and large scale (forest surveying, satellite mapping). Many of this solutions use photogrammetry which is the science that focuses on the inference of physical properties of objects such as position and shape from the use of photographic images and other means of electromagnetic imaging (such as LIDAR).

One remarkable instance of 3D mapping problems with vehicles are the problems known as SLAM (Simultaneous Localization and Mapping) which use sensors to construct a 3D map of the environment in order to position themselves while being aware of potential hazards in the environment. The main objective in this problems is to achieve the positioning and awareness, but they can produce useful maps. One distinctive solution in the SLAM field (called monoSLAM) manages to create voxel-based maps with color using a single camera.

Some of the most direct commercial solutions use the power of drones for higher detail mapping. There exists a commercial LIDAR drone (the Elios 2) capable of generating very high-detail 3D models (around 1mm) of the environment but at the expense of being heavy, big and costly. There are also lower end mapping drones as well as specific software and professional services for this kind of mapping.

LIDAR is a technology that is being used in UAVs for high precision distance measurements. It can be seen as a more precise but lower range alternative for radar that can perceive the shape of solid objects through the emission of light pulses. If the pulse bounces back to the sensor, it can measure the time taken since the emission of the pulse and determines the 3D point where the pulse bounced. LIDAR can thus accurately model shapes with the emission of up to millions of pulses per second.

Although most of the market is in heavier and better performing LIDAR sensors, there are lightweight sensors that can be attached to small drones with fairly good resolution (hundreds of thousands of points per second) but lower range (many under 100m), and with the added issue of precision problems from the positioning sensors.

For high precision LIDAR to be possible in small UAVs without the use of complex GNSS systems, there needs to have improvements in the technologies involved in the position of the drone (IMUs and GNSS) so that the error in the positioning of the drone doesn't propagate to the LIDAR data. Thankfully, there have been consistent advancements in this technologies in the last decades and it could be expected to become only a minor problem in the future.

The last topic of interest for this project is autonomous navigation of UAVs. It has been a topic of interest for the last decades but only took off recently thanks to advances in artificial intelligence, with the most notorious case probably being autonomous driving in cars using machine learning.

With problems ranging from moving from point A to point B, to complete exploration of environments while dodging obstacles, autonomous navigation has seen a lot of advances. Currently there are state of the art solutions such as autonomous search and rescue navigation algorithms in drones for unknown environments.

For complex navigation problems machine learning's deep learning algorithms stands out as a very flexible and powerful solution that can find strategies for pro-

blems that would otherwise be nearly impossible to accurately solve for humans. Particularly, deep reinforcement learning has proven to work for problems such as the exploration of unknown terrains.

State of the art: Reinforcement learning

As a product of the complexity of the problem of autonomous navigation in this case, most simple approaches are quickly ruled out but not without leaving some hope in the field of reinforcement learning. The branch of machine learning is then investigated in order to be able to build a proper approach to this problem using its algorithms.

In general, a bet is put in reinforcement learning because it has demonstrated good results and generalization in very complex problems like drone exploration of unknown areas and even showed outstanding results such as the beating of professional players in complex video games like Dota 2 (even though this was only possible after months of training in a supercomputer).

First, the bases are investigated. The most standard approach to reinforcement learning divides the problem in two parts: the environment and the agent. The environment is the element that is being attempted to solve by the agent which interacts with it while learning in order to slowly improve.

Although the environment can represent any problem, it is common to establish a standard interface for agents to interact with it. This interface should be able to, at a specific time, execute the actions that the agent specifies and then provide the required information from the environment which is composed of an observation of itself (that should accurately represent its state), a value of reward, and a condition of termination specifying if the instance of the problem has ended.

The environment and its workings have a formal mathematical framework called Markov Decision Process (and its existing variations) which is useful to define problems accurately and how they should be approached. These MDPs are the base of most frameworks for the implementation of reinforcement learning algorithms such as OpenAI gym.

The reward value given by the environment after every action represents the performance of the execution. If a goal is achieved, an environment will generally provide a positive reward, but if something unwanted occurs, a negative value will be provided instead. The goal of reinforcement learning is to maximize the accumulated reward value after many steps, which is formally called the return of the trajectory. An agent will thus seek high rewards and avoid low ones.

The reward function in a MDP $R_a(s, s')$ is the function that determines the reward value after an action a is taken given the state transition s to s'. A good reward function should consistently provide positive reward when the agent approaches the

goal and negative when it wards off from it. Although a simple sounding task, creating a good reward function can be one of the hardest and most critical steps of designing a good reinforcement learning algorithm.

The goal of the agent should be to learn in order to maximize the total reward gained in the long term. The learning occurs in the policy which is the function that maps the observation space to the action space, or in other words, it's the logic that chooses what action to execute given an observation.

To update the policy in order to perform the learning, a learning algorithm (or optimizer) is required. This is where a choice of design appears as there is a big selection of algorithms which perform differently in distinct circumstances.

The learning algorithms branch off in various ways, but for general purpose algorithms, the most key distinction in the learning is whether they use what is called a value function, or the direct modification of the policy.

The value function is a function that essentially specifies how "good" it thinks a situation is. This is done by directly trying to predict the (discounted) accumulated reward that the best known set of actions could give. Some algorithms, particularly Q-Learning and its many variations learn, in a nutshell, by trying to model a value function and then using it to choose the action that it predicts is better. The modelling of this function occurs through a process called temporal difference learning.

A notable variation of the use of a value function are called "actor-critic" and instead of directly choosing the action that is predicted best, the decision comes from a separated policy. This learned value function is then used to "critic" the choices of the policy and learn from it.

Many types of algorithms are evaluated but it is decided in this case to go for Deep Reinforcement Learning algorithms as they have shown a good level of generalization and flexibility. This algorithms use artificial neural networks, which are universal function approximators, to learn the policy and/or the value function. Particularly an actor-critic learning algorithm named PPO is selected.

Design: mapping tool

The first part of the system consists in the algorithm and data structures involved in the construction of the 3D map. Because small drones can't carry devices with high computing capability, it is important to keep a good performance in the algorithm. A low time complexity algorithm will also help lower the training time which can take up to several days per agent.

LIDAR, being the main mapping sensor, generates large amounts of data that need to be processed and stored. The reinforcement learning environment will per-

form computations on the constructed map, so to get a good performance the tool generates an auxiliary low resolution map in a tree based structure to allow for quick insertion and search of data. The redundant data (which is most of it) is deleted in order to keep the map fast. Alongside this low resolution map then the final LIDAR data can be stored with minimal processing, or alternatively, a higher resolution tree based structure can be used.

Fortunately a framework prepared for these cases already exists and is called OctoMap. OctoMap uses a tree-type data structure called octree to represent three-dimensional space using voxels, and can simultaneously represent occupied, free and unknown space. The search and insertion complexity of this algorithm is of $O(n) = log(n)$, which is very fast, and automatically ignores data that coincide with already modeled space.

The tool is essentially a high-level wrapper for OctoMap with some extras. The final tool then takes points produced by LIDAR and builds this voxel map while storing the raw LIDAR data for later processing. It also allows the set up of bounds for the map where all generated points outside of them will be ditched.

Although the original OctoMap implementation already includes visualization tools, for compatibility issues a new simple visualization tool is implemented and attached to the mapper. This visualization tool extracts the data in the map and uses the mayavi python library for the 3D visualization.

This tool could be, with some modifications, be used in real drones to construct maps of lower or higher quality. To be of any use though, the drone would need to send the visualization data to the user while in the air so the user can know the progress of the mapping.

Design: environment

For the drone and lidar, as explained before a simulator is used. The simulator of choice is AirSim which is the go-to drone simulator for machine learning applications (but can also simulate other vehicles). AirSim allows for a lot of customization and possesses a complete and flexible API for the control of the drone and the environment. A key feature of AirSim for this work is that it also allows the simulation of LIDAR sensors with a wide range of configurations.

In order to use a reinforcement learning algorithm, it is necessary to design first its environment. This environment must wrap its inner working logic (in this case, AirSim), and then provide an standard interface for the agent to use.

The environment interface must implement observations, a reward function, a termination condition and a way to execute drone movement from abstract actions. Also, a reset functionality is necessary for the training.

Many parts of the environment are critical to the proper learning of the agent and require a well though design. For example, an inaccurate or partial observation could prevent the algorithm from converging to a good policy.

Starting with the environment perception, it is important to ensure that the information given is as complete and representative as possible about the state of the environment. It is also useful to keep the observation space size small to help convergence.

To give a complete observation, it is decided to divide the perception in two parts. The first is a depth based image of the vision of the drone in 360^{o} horizontally to give information of the potential obstacles and hopefully, to help localize patterns for local exploration such as holes to be explored. This depth image will be composed from the constructed OctoMap using ray-casting in the voxel space, so this way an additional camera won't be needed. Because the maximum depth of the image is higher than the LIDAR range unreached areas could be perceived, which could help identifying further unexplored areas. The resolution is kept low at around 100x40 pixels.

The second part of the perception is a top-view 2D map of the explored and unexplored areas, which is fed to the agent as a flat vector. This map composed of a few quadrants, represents how much time the drone has spent in every quadrant with values from 0 to 0.5 and then in which position the drone is with a value of 1. This is made to give information about the global state of the exploration and so the agent can know which areas are yet to explore.

Then, the actions that the agent can perform are defined. Two different sets of discrete actions are proposed to test separately. First, a more complex but more flexible approach is defined where the agent can accelerate in any direction a small value per action. This allows for the agent to accelerate as much as it wants and thus explored at the desired speed, but at the cost of added complexity as the agent must also learn to control its speed which, for non-recurrent architectures, could require the use of techniques like frame-stacking in order for the agent to be aware of its speed.

The other proposed set of actions is defined by allowing the agent to directly move in any direction a set amount of distance. This is far less complex and thus easier to learn, but at the cost of less flexibility and the inability for finer maneuverability which could be necessary to explore certain areas.

Also, for the training, a termination condition and a reset function has to be defined. The proposed termination condition basically requires the agent to achieve a minimum number of discovered area. If the agent doesn't find any new area for a set number of steps, the termination condition is triggered. This condition ensures that the training is kept dynamic and also guarantees termination when the map is completed as there would be no more areas to explore. A collision of the drone with

the environment will also trigger the termination condition.

To allow for generalization, and thus allowing the exploration of unknown environments, the reset function changes the exploration area to a different one every time. In AirSim this can be done in big environments by changing the exploration bounds to a completely different area for a number of episodes, and then, after a number of changes, completely change the world by closing the AirSim process and opening a different one. This changes are done randomly.

Finally, the reward function of the environment is defined. As this is one of the hardest elements to properly define, two different rewards are proposed. The first one, being the most dynamic, rewards the agent for every number of discovered voxels in the map every step up to a fixed number. This reward can encourage a higher exploration speed but it has a set ceiling to control how local (and slow) the exploration should be. The main problem of this reward function is that it is prone to create flat areas in the reward value space that can lead to the stagnation of the reward, and as is explained later, this is what happens.

The second reward function is based on the most guaranteed method for its design which is called potential-based reward shaping. This reward given is then proportional to "how close" the state is to the goal and ignores the specific details of the last transition. In this case, the goal is the creation of a complete map which in OctoMap, could be represented by the total amount of voxels that compose a complete map. Because the total amount of voxels in a complete map is unknown a general guess has to be made and expect to not undershoot it. The reward function then gives a reward value proportional to how close it is to the goal, in this case, it's defined by the function $r = n_{voxels} \cdot C$, with C being a predefined constant. The main problem with this reward is that it cannot control the speed of exploration and could lead to very slow or very fast exploration depending on the agent configuration.

Apart from that, for both cases a strong negative reward will be given in case of collision and another mild negative reward in case of the drone wandering outside of the exploration bounds.

Design: autonomous navigation

Now a policy and a learning algorithm have to be chosen. As explained earlier, deep reinforcement learning is chosen for its generalization capabilities and flexibility. For the learning algorithm (or optimizer) PPO, an actor-critic algorithm, is selected for being one of the known best optimizers.

Because PPO is a DRL algorithm, it requires a neural network architecture. Being of the actor-critic class specifically, it requires two neural networks, one for the value function, and another for the policy. It is also necessary to realize that because the observation is made from two different types of input (a vector and an

image), a split architecture for the input has to be used.

For this, two architectures are proposed. The first being of the feed forward type, and the second of the recurrent type, which is a common practice to compare these two types. Because training takes very long in this case, it is a necessity to minimize training steps. To do this, what is called the feature extractor (which are essentially the first layers) is shared for both the value function and the policy. This is a practice that has been proved to work in other problems.

The feed-forward type is composed from proven architecture components for both inputs and outputs and is the default actor-critic multi-input architecture in the library stable baselines 3 which shares the feature extractor layers. This architecture uses convolutional layers and pooling for the early processing of the image input and fully connected layers for the rest.

The recurrent architecture is built completely in PyTorch using an additional single LSTM layer at the end of the shared feature extractor. The rest of the architecture is kept similar to the feed-forward but a bit more complex. The PPO optimizer is also built from scratch which might perform differently as others as there are different interpretations as how it should be implemented.

Results of the autonomous navigation

As stated before, training takes up to days in this environment, with the problem being that many configurations of the environment have to be tested with limited time and also having to fine-tune the hyperparameters of the architectures.

The first configuration is tested with the acceleration-based actions and the first reward function making it the most complex configuration. Unsurprisingly, this configuration shows the worst results.

The second configuration uses the fixed movement set of actions and performs slightly better with a specific configuration of the recurrent architecture performing slightly better. Still, the results are underwhelming and after some analysis, it is determined that the given reward function is too inconsistent with a topology insufficiently smooth and consistent in its reward space, leading always to essentially the same local maximum.

Then the second reward function with the second set of actions is tried and the feed-forward architecture, showing initially better results. Unfortunately, time runs out before having sufficient testing with this set up. But still, it might not have shown good enough results to be of use.

None of the trained agents are capable of achieving an useful performance and all of them converge towards some level of local exploration while ignoring big parts of the map. Although the second reward function showed some promise, it is likely

that it wouldn't achieve sufficient results either.

Conclusions

The performed investigation of the relevant technologies show that in the future it might be possible to use LIDAR in lightweight drones for high precision three-dimensional mapping. Also the current research in autonomous navigation of UAVs point towards the possibility of developing an autonomous navigation system for this LIDAR mapping drone.

For the development part, this work finds, tests and combines key software and ideas that can be of use for related work such as a better solution for the problem or for similar problems.

About the autonomous agent, it is concluded that while reinforcement learning could still be capable of solving the problem, a different approach to the one taken could be necessary. Maybe it is as simple as changing one of the environment elements, or maybe a more hierarchical approach is necessary.

Another option is that this is directly not a viable problem for reinforcement learning. It is known that reinforcement learning algorithms usually require disproportionate amounts of data in training to achieve good generalization. Maybe this problem could be properly solved with more complex architectures trained in sufficiently powerful computers and longer training times.

ANEXO B. LISTA DE ABREVIATURAS

A2C	Advantage Actor Critic
ACER	Sample Efficient Actor-Critic with Experience Replay
API	Application Programming Interface
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient (algoritmo)
DDQN	Double Deep Q-Network (algoritmo)
DQN	Deep Q-Network (algoritmo)
DRL	Deep Reinforcement Learning
DRQN	Deep Recurrent Q-Learning
EU	European Union
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IA	Inteligencia Artificial
IEEE	Institute of Electrical and Electronics Engineers
IMU	Inertial Measurement Unit
ISO	International Organization for Standardization
LIDAR	LIght Detection And Ranging
LSTM	Long Short Term Memory
MDP	Markov Decision Process
NASA	National Aeronautics and Space Administration
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization (algoritmo)
RGB	Red Green Blue
RL	Reinforcement Learning
SLAM	Simultaneous Localization And Mapping
TDL	Temporal Difference Learning
TFG	Trabajo de Fin de Grado
TRPO	Trust Region Policy Optimization (algoritmo)
UAV	Unmanned Aereal Vehicle
UC3M	Universidad Carlos III de Madrid
VANT	Vehículo Aéreo No Tripulado