UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Datos heterogéneos aplicados al entrenamiento de un algoritmo de planificación de rutas basado en aprendizaje por refuerzo

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: López Muñoz, Pablo

Tutor/a: Monserrat del Río, José Francisco

Director/a Experimental: PALACIOS MOROCHO, MARITZA ELIZABETH

CURSO ACADÉMICO: 2022/2023

Resumen

El constante desarrollo de la Inteligencia Artificial ha propiciado que esta se haya convertido en una de las principales tecnologías existentes en la actualidad, debido al amplio abanico de beneficios que ofrece tanto a la sociedad como a la industria y los negocios. Muchas son las ramas que tiene, siendo una de las principales la conocida como Machine Learning, que está consiguiendo importantes avances en el desarrollo de robots autónomos, gracias a algoritmos de Reinforcement Learning.

Este trabajo de fin de grado pretende continuar con el desarrollo de este campo, focalizando el trabajo en el tratamiento de datos heterogéneos para el entrenamiento de dos redes neuronales, que se encargarán de entrenar a robots mediante un algoritmo de Reinforcement Learning, con el fin de que estos puedan alcanzar diferentes metas a largo de un entorno establecidos. Para alcanzar este objetivo se trabajará a partir de un algoritmo base, con el fin de conseguir que el robot funcione correctamente independientemente del sensor que proporcione los datos a las redes neuronales, mediante la transformación de datos heterogéneos a un mismo formato común, compatible con estas redes neuronales.

Por otra parte, también se trabajará para que el robot sea computacionalmente lo más sencillo posible, descentralizando ciertos procesos fuera del robot, de forma que el entrenamiento de las redes neuronales sea llevado a cabo en un servidor externo, que estará constantemente intercambiando información con el robot. Esto se llevará a cabo mediante la técnica de paralelización con la que se consigue utilizar diferentes núcleos del CPU en paralelo, de esta forma se libera a este de la carga computacional del entrenamiento, y se reducen los procesos que debe realizar.

Palabras claves

Reinforcement Learning, conducción autónoma, datos heterogéneos, pointcloud_to_laserscan, paralelización.

Abstract

The constant development of Artificial Intelligence has meant that it has become one of the main technologies actually, due to the wide range of benefits it offers to society, industry and business. It has many branches, being one of the main ones known as Machine Learning, which is making important advances in the development of autonomous robots, thanks to Reinforcement Learning algorithms.

This final degree project aims to continue with the development of this field, focusing the work on the treatment of heterogeneous data for the training of two neural networks, which will be responsible for training robots by means of a Reinforcement Learning algorithm, so that they can achieve different goals throughout an established environment. To achieve this objective, I will work from a base algorithm, in order to get the robot to work correctly regardless of the sensor that provides the data to the neural networks, by transforming heterogeneous data into a common format, compatible with these neural networks.

On the other hand, I will also work to make the robot as computationally simple as possible, decentralising certain processes outside the robot, so that the training of the neural networks is carried out on an external server, which will be constantly exchanging information with the robot. This will be done using the technique of parallelization, which allows different CPU cores to be used in parallel, this relieves the robot of the computational burden of training and reduces the number of processes it has to perform.

Resum

El constant desenvolupament de la Intel·ligència Artificial ha propiciat que aquesta s'haja convertit en una de les principals tecnologies existents en l'actualitat, a causa de l'ampli ventall de beneficis que ofereix tant a la societat com a la indústria i els negocis. Moltes són les branques que té, sent una de les principals la coneguda com aprenentatge màquina, que està aconseguint importants avanços en el desenvolupament de robots autònoms, gràcies a algorismes de aprenentatge reforçat.

Aquest treball de fi de grau pretén continuar amb el desenvolupament d'aquest camp, focalitzant el treball en el tractament de dades heterogènies per a l'entrenament de dues xarxes neuronals, que s'encarregaran d'entrenar a robots mitjançant un algorisme de aprenentatge reforçat, amb la finalitat que aquests puguen aconseguir diferents metes a llarg d'un entorn establits. Per a aconseguir aquest objectiu es treballarà a partir d'un algorisme base, amb la finalitat d'aconseguir que el robot funcione correctament independentment del sensor que proporcione les dades a les xarxes neuronals, mitjançant la transformació de dades heterogènies a un mateix format comú, compatible amb aquestes xarxes neuronals.

D'altra banda, també es treballarà perquè el robot siga computacionalment el més senzill possible, descentralitzant uns certs processos fora del robot, de manera que l'entrenament de les xarxes neuronals siga dut a terme en un servidor extern, que estarà constantment intercanviant informació amb el robot. Això es durà a terme mitjançant la tècnica de paral·lelització amb la qual s'aconsegueix utilitzar diferents nuclis del CPU en paral·lel, d'aquesta manera s'allibera a aquest de la càrrega computacional de l'entrenament, i es redueixen els processos que ha de realitzar.

Índice general

Glosario de términos

	Description (ING)	**Descripción (ESP)**
(API)	Application Programming Interface	Interfaz de programación de aplicaciones
(CPU)	Central Processing Unit	Unidad central de procesamiento
(CNN)	Convolutional Neural Network	Redes Neuronales Convolucionales
(DDQL)	Double Deep Q-Learning	Doble aprendizaje Q profundo
(DQL)	Deep Q-Learning	Aprendizaje Q profundo
(DRL)	Deep Reinforcement Learning	Aprendizaje por refuerzo profundo
(GPU)	Graphics Processing Unit	Unidad de procesamiento gráfico
(IA)	Artificial Intelligence	Inteligencia artificial
(LiDAR)	Light Detection and Ranging	Detección y localización de la luz
(MCU)	Uniform Circular Motion	Movimiento Circular Uniforme
(ML)	Machine Learning	Aprendizaje automático
(MPI)	Message Passing Interface	Interfaz de paso de mensajes
(NLP)	Natural Language Processing	Procesamiento del lenguaje natural
(ODS)	Sustainable Development Goals	Objetivos de desarrollo sostenible
(Q-L)	Q Learning	Aprendizaje Q
(RIC)	Interquartile range	Rango Intercuartílico
(ReLU)	Rectified Linear Unit	Unidad Lineal Rectificada
(RGB)	Red Blue Green	Rojo verde azul
(RL)	Reinforcement Learning	Aprendizaje de refuerzo
(ROS)	Robot Operating System	Sistema Operativo Robótico
(SLAM)	Simultaneous Localization and Mapping	Localización y mapeo simultáneos
(5G)	Fifth Generation	Quinta generación

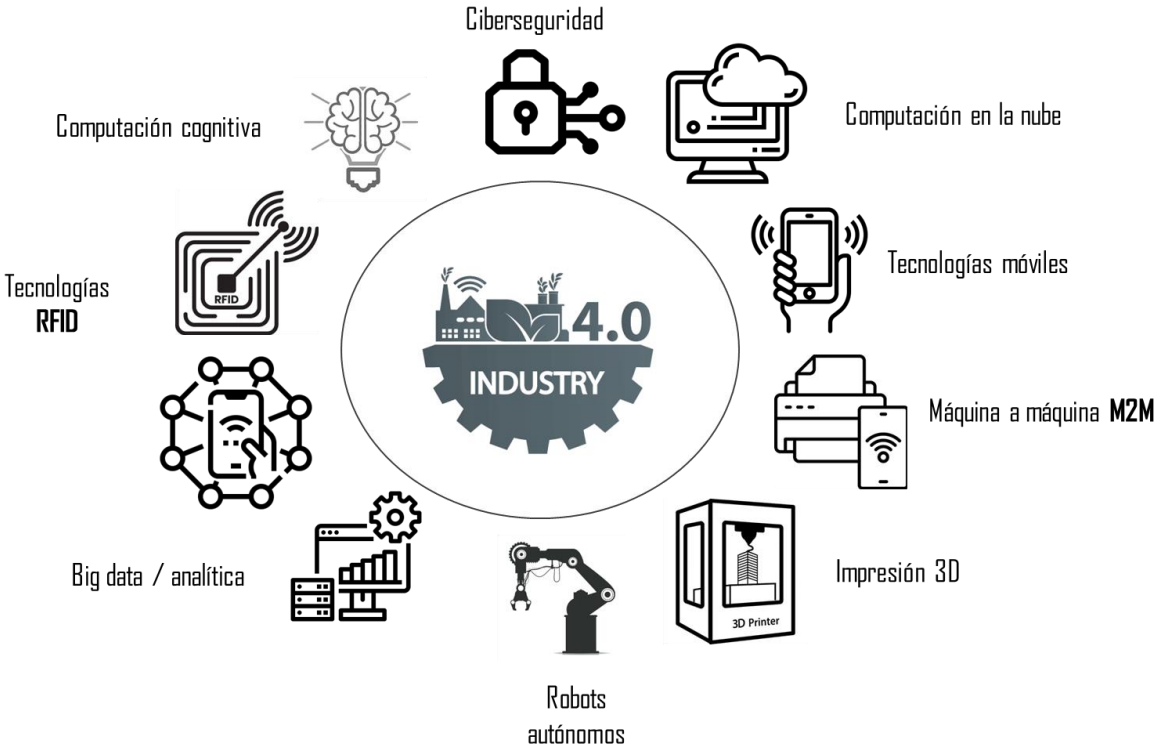
Capítulo 1. Introducción

1.1 Contexto

A lo largo de la historia el desarrollo tecnológico ha intentado ir de la mano del desarrollo humano y de las necesidades sociales que surgen constantemente, algunas veces ha sido la tecnología y sus avances lo que ha impulsado el desarrollo de la sociedad, como pueden ser todos los avances tecnológicos que se realizan en las zonas menos desarrolladas, como el continente africano, donde la tecnología se muestra como un medio para conseguir un desarrollo social. Otras veces, en cambio, es el contexto histórico y social el que impulsa a la tecnología desarrollarse, como sucedió, por ejemplo, con la pandemia del coronavirus, en la que la tecnología tuvo que adaptarse a las necesidades de la sociedad cuando fue necesario su uso para mantener la actividad económica y social a pesar de los confinamientos y restricciones.

Sea de una manera u otra, hay un aspecto que es común, y es que el desarrollo tecnológico siempre ha desencadenado una serie de desafíos y oportunidades, modificando sustancialmente los paradigmas de la sociedad, de la economía y de la industria. En relación a esta última, la tecnología ha conseguido transformaciones sin precedentes en los procesos de producción, y es que desde el siglo XVIII, en el que se produce la primera revolución industrial, hasta la actualidad, la industria ha sufrido cambios tan importantes como fueron la mecanización de procesos en la primera revolución industrial gracias a la energía de vapor, el uso de la electricidad y la producción en cadena en la segunda revolución industrial, la aplicación de la informática en el proceso productivo en la tercera revolución consiguiendo una mayor automatización y finalmente, la cuarta revolución industrial, que trabaja en la digitalización de la industria [1].

Figura 1 Tecnologías presentes en la industria 4.0.



En la Figura 1 se pueden apreciar algunos de los diferentes avances tecnológicos que se han producido en esta cuarta revolución industrial, y es que en la última década con la llamada “industria 4.0”, han sido gigantescos los avances tecnológicos, debido a su gran potencial y a los beneficios relacionados

con la integración, innovación y autonomía de los procesos. El impacto de esta transformación tecnológica es tal, que aquellas empresas capaces de adoptar estas tecnologías emergentes han visto como gran parte de los aspectos de la organización, desde la organización y producción, hasta el desarrollo e investigación, han mejorado considerablemente gracias a la aplicación de los avances tecnológicos. Estos están encaminados principalmente a generar una red de producción inteligente, en la que mediante la colaboración e interacción entre sistemas se pueda flexibilizar, automatizar e individualizar diversos procesos productivos e incrementar la eficiencia de estos [2].

De estos avances tecnológicos que aprecian en la Figura 1 es importante resaltar el del uso de robots autónomos en la industria, debido a que este trabajo de fin de grado está enfocado a ellos. Esta rama de la industria 4.0 se puede destinar a realizar diversas actividades, entre las que destaca la navegación autónoma, en la que sistemas robóticos autónomos, estando en entornos desconocidos para ellos, deben ser capaces de tomar decisiones y llevarlas a cabo con autosuficiencia, es decir, sin necesidad de control o intervención humana, para alcanzar ciertas metas u objetivos establecidos previamente.

El uso de la navegación autónoma mediante robots es cada vez más común en la actualidad, debido a los beneficios y oportunidades que ofrece y a la cantidad de aplicaciones que pueden tener en la sociedad y la industria, es por esto por lo que en este trabajo de fin de grado se ha querido profundizar sobre esta rama. Por ello, se analizará de qué forma influyen los datos que necesita el robot para funcionar, debido a que, en la navegación autónoma, los robots necesitan constantemente información del entorno que los rodea, que obtienen a partir de distintos sensores que tienen incorporados, para poder tomar la mejor decisión sobre qué acción realizar en cada momento, para poder alcanzar sus metas.

De esta forma, en este trabajo de fin de grado se trabajará con distintos sensores que puede utilizar un robot para, a través de un algoritmo de Reinforcement Learning (RL), alcanzar sus metas. Con ello, se trabajará con datos heterogéneos, ya que cada sensor aportará datos distintos, con el fin de adecuar estos datos de forma que todos presenten el mismo formato y puedan ser utilizados por el robot de forma indiferente, consiguiendo que el robot funcione correctamente independientemente de que utilicen un sensor u otro.

1.2. Objetivos

El objetivo principal de este trabajo de fin de grado es adaptar los datos procedentes de sensores heterogéneos para poder ser utilizados en un algoritmo de planificación de rutas mediante RL que requiere de datos homogéneos.

Para conseguirlo, se seguirán los siguientes objetivos específicos:

- Conocer los conceptos generales y el funcionamiento de los algoritmos de Machine Learning (ML) y especialmente los de RL.
- Familiarizarse con la simulación robótica mediante el software de ROS y su entorno de simulación Gazebo.
- Incorporar distintos sensores en el robot y estudiar las características de los datos heterogéneos procedentes de cado uno de ellos.
- Procesar los datos obtenidos de fuentes heterogéneos con el fin de convertirlos todos al mismo formato, para poder ser utilizados en el algoritmo original.
- Filtrar y sincronizar los datos cuando se utiliza más de un sensor simultáneamente.
- Realizar un proceso de relleno de datos en aquellos casos que los datos proporcionados por el sensor son menos de los requeridos por las redes neuronales.
- Conseguir que el robot sea computacionalmente lo más sencillo posible consiguiendo externalizar funcionalidades a una nube/servidor que realice las funciones más costosas, de forma que el robot sea económico.

- Entrenar una nueva política a partir de la original para determinar la influencia de los datos heterogéneos.
- Comparar el rendimiento de las nuevas políticas obtenidas con los nuevos sensores instalados y con el sensor original, realizando diversas pruebas.

1.3. Motivación

En el mundo actual, el día a día va cambiando dado a la presencia de la inteligencia artificial (IA) en muchas de los campos tanto de la industria como de la sociedad. La planificación de rutas con vehículos autónomos son uno de los principales campos de estudio de la IA, ya que representan un reto en términos de fiabilidad y seguridad, a la vez que se muestran como uno de los principales avances tecnológicos en el futuro más inmediato gracias a los múltiples beneficios y oportunidades que aportan. En la actualidad existen muchos algoritmos de planificación de rutas, pero estos están enfocados en el uso de sensores homogéneos, ya que lograr la compatibilidad entre sensores heterogéneos representa un reto.

Esta ha sido la principal motivación para llevar a cabo este trabajo de fin de grado, el poder profundizar en un campo tan importante hoy en día como es el de los vehículos autónomos y además poder trabajar en un aspecto que no está tan desarrollado como es el uso de sensores heterogéneos en redes neuronales homogéneas. Para ello el trabajo se ha centrado en los datos que aportan los distintos sensores, en contenido y estructura, para ver de qué forma es posible adaptarlos, así como sincronizarlos, para aquellos casos que existe más de un sensor integrado en el robot. Además, también ha supuesto un reto, como aproximar y rellenar datos de forma precisa, en aquellos sensores que aportan menos datos de los requeridos para hacer funcionar la red neuronal, como sucede con los datos de la cámara.

Por último, pensando siempre en una posible implementación física del trabajo en el futuro, se ha buscado en todo momento que el coste de llevarlo a cabo fuese el menor posible. Por lo que conseguir que el robot sea lo más económico posible, a través de conseguir que sea computacionalmente lo más sencillo posible externalizando funcionalidad a la nube, ha sido también una motivación muy importante.

1.4. Principales contribuciones

Las principales contribuciones de este trabajo de fin de grado están directamente relacionadas con los objetivos planteados para el mismo, y se pueden sintetizar en:

- Homogenización y adaptación de la salida de datos procedentes de sensores heterogéneos que permitan el uso de los mismos en algoritmo de planificación diseñado para sensores homogéneos, que tengan una misma estructura y formato de datos de entrada. Es decir, compatibilidad entre robots heterogéneos usando un algoritmo homogéneo.
- Dado que la entrada de datos de la red neuronal es estándar y que cada sensor tiene un numero de datos de salida diferente, se diseñó un algoritmo que permite la aproximación de datos para aquellos sensores cuya salida es menor que la requerida por la red neuronal. La aproximación de datos permite estimar la distancia del robot hacia diferentes puntos ya conocidos sin la necesidad de tener un sensor midiendo los mismos.
- Sincronización de datos entre diferentes sensores.
- Reducción de la capacidad computacional que requiere el robot para funcionar, a raíz de mover funcionalidades a una nube/servidor externo.
- Desarrollo de una arquitectura centralizada que decide la acción a realizar por el robot, de forma que este solo tiene que realizarla, lo que implica que el ordenador encargado de las funciones del robot no necesita tanta computación.
- Lograr que un robot con un sensor de menor tecnología, como una cámara, tenga un comportamiento igual al de un robot con sensores de alta tecnología, como un láser.

- Reducción de precios de despliegue de estos servicios dado que el robot no necesita sensores de alta tecnología ni ordenadores con alta capacidad computacional para procesar los datos, dado que el robot solo obtiene los datos del entorno a partir de los sensores y ejecuta las acciones enviadas por la nube.

1.5 Organización de la memoria

En esta sección se explica cuál va a ser la organización y estructura de la memoria, explicando cada uno de los capítulos que la integran, que son los siguientes.

Capítulo 2. Estado del arte. En este primer capítulo se explica cuál es la situación actual de las tecnologías que se utilizan en el trabajo, empezando por la que las engloba todas, la IA, para después ir analizando cada uno de los campos que sirven de base al algoritmo desarrollado y utilizado.

Capítulo 3. Tecnologías utilizadas. Esta parte tiene la función de explicar tanto el hardware como el software que se ha utilizado para el desarrollo del trabajo, explicando desde el lenguaje de programación principal, Python, hasta las librerías que han sido más importante para poder implementar cada una de las funciones a llevar a cabo por el robot y la nube.

Capítulo 4. Metodología. Esta es la parte principal y es donde se explica todo el proceso de homogeneización, sincronización y relleno de los datos heterogéneos. En primer lugar, se explica cómo se ha conseguido separar las funcionalidades entre agente y nube, después se procede a mostrar cuál ha sido el robot elegido, así como los sensores que se utilizarán, explicando sus características y los datos que aportan. A continuación, se explicará cómo se ha llevado a cabo tanto la homogeneización y sincronización de los datos, para finalizar explicando cómo se ha procedido al relleno de datos en aquellos casos que ha sido necesario.

Capítulo 5. Pruebas y resultados. En este capítulo es en el que se exponen los resultados de las distintas pruebas que se han realizado, analizando el entrenamiento de cada una de las configuraciones de los sensores en los distintos entornos donde se ha realizado, así como se analizará el comportamiento de algunas configuraciones cargando en ellas el entrenamiento realizado con otra configuración distinta.

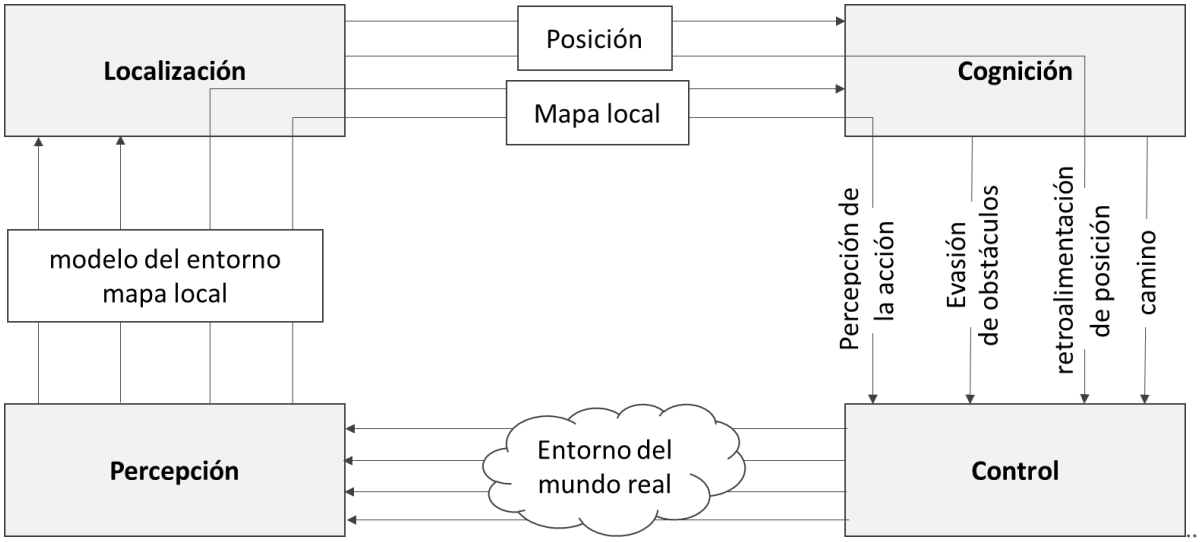
Capítulo 6. Conclusiones y líneas futuras. En el último capítulo se hará una recapitulación del trabajo realizando, extrayendo las conclusiones más importantes del mismo. Así mismo también se expondrán las posibles mejoras futuras en las que se podría trabajar a fin de profundizar aún más en este tema.

Capítulo 3. Estado del Arte

Como se ha visto en la introducción y en la Figura 1, muchos son los avances que se están presentando en la actualidad en relación con la digitalización de la industria gracias a la aplicación la aplicación de la inteligencia artificial, la inteligencia de datos o el desarrollo de redes móviles como es la quinta generación para redes celulares (5G), entre otros. Todos esto ha provocado diversos avances, entre los que destaca la navegación autónoma, clave en este trabajo de fin de grado, y es que, desde hace varios años, la navegación autónoma mediante robots se ha presentado como uno de los mayores retos tecnológicos.

Muchos son los procesos, y complejos, que debe seguir un robot para desplazarse de forma autónoma eficientemente, pero a nivel general, para entender el funcionamiento global de un robot autónomo, este debe ser capaz de realizar una serie de acciones y procesos correctamente para poder alcanzar sus objetivos, que quedan sintetizados en la Figura 2.

Figura 2 Acciones que debe ser capaz de realizar el robot para poder tener éxito en la navegación autónoma.



En la Figura 2 se puede observar claramente cuatro bloques [3]:

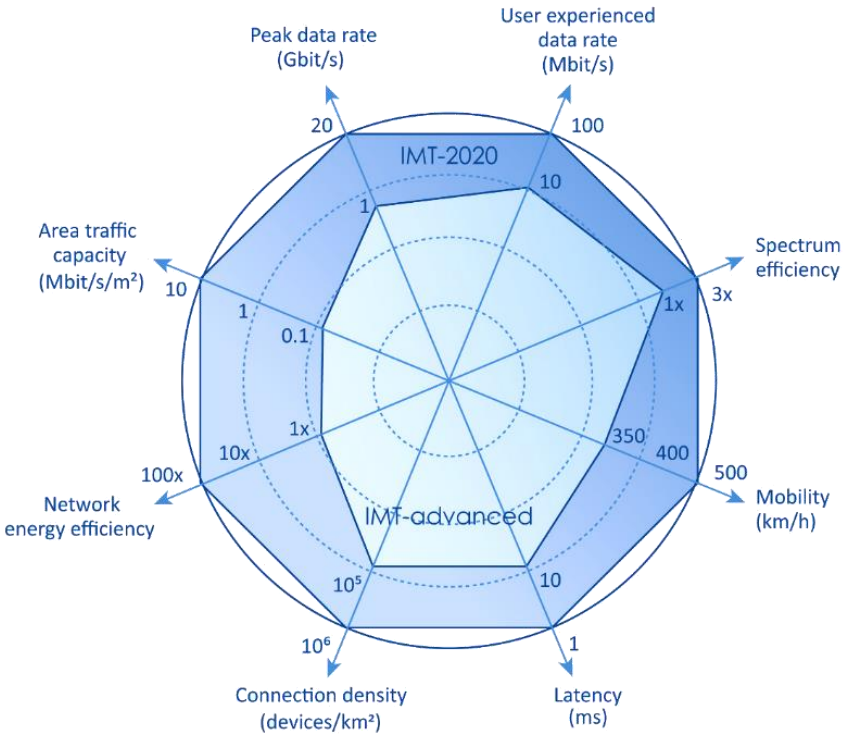
- Percepción: El robot debe ser capaz de extraer datos útiles a través de la interpretación de los diferentes sensores (laser, cámara, etc.) que pueda llegar a tener incorporados.
- Localización: El robot debe ser capaz de determinar su posición en el entorno en el que se encuentra.
- Cognición: El robot debe ser capaz de tomar una decisión a partir de los datos extraídos con los sensores y su posición en el entorno, para la consecución de manera eficiente de su objetivo, así como ser capaz de responder a ciertas situaciones imprevistas y evadir obstáculos inesperados que pueda encontrar en su camino.
- Control: El robot debe ser capaz de ejecutar de manera automatizada las decisiones que ha tomado, con el fin de desplazarse con la trayectoria deseada y alcanzar su objetivo marcado de la forma más eficiente posible.

Estos son los 4 bloques, o acciones que tiene que ser capaz de llevar a cabo un robot autónomo para funcionar correctamente, alguna de ellas se desarrollará más en profundidad a lo largo del trabajo, como la referida a la percepción, con la extracción de datos útiles de los sensores o la cognición, en relación con el método de toma de decisión del robot. Por otra parte, la mejora de las capacidades de las redes móviles también ha sido un factor muy importante en el desarrollo de los robots y vehículos autónomos,

debido a que con las nuevas tecnologías en comunicaciones se han conseguido alcanzar gran parte de los requerimientos necesarios para el funcionamiento de robots autónomos.

La tecnología 5G ha sido la que ha permitido alcanzar los estándares necesarios, ya que con ella se ha conseguido aumentar la capacidad de redes móviles para garantizar comunicaciones que permitan que la conducción autónoma sea fiable, segura y eficiente. Alguna de las características de 5G que ha permitido esto son el aumento del ancho de banda, aumento de la capacidad de la red o la comunicación ultra confiable gracias a la baja latencia que hay con esta tecnología. En la Figura 3 se muestra una comparación de las capacidades de las redes móviles 5G en relación con la tecnología anterior existente, las redes 4G [4].

Figura 3 Comparación entre las capacidades de las redes 4G y 5G [5]



Finalmente, el pilar fundamental en el cual se sustentan los robots autónomos es la IA, ya que gracias a esta se consigue que los robots puedan capturar la información del entorno, analizarla y tomar decisiones de forma autónoma. En los siguientes apartados se comentará en que consiste la IA, así como las distintas ramas en las que se divide, para poder entender los algoritmos utilizados en este trabajo de fin de grado.

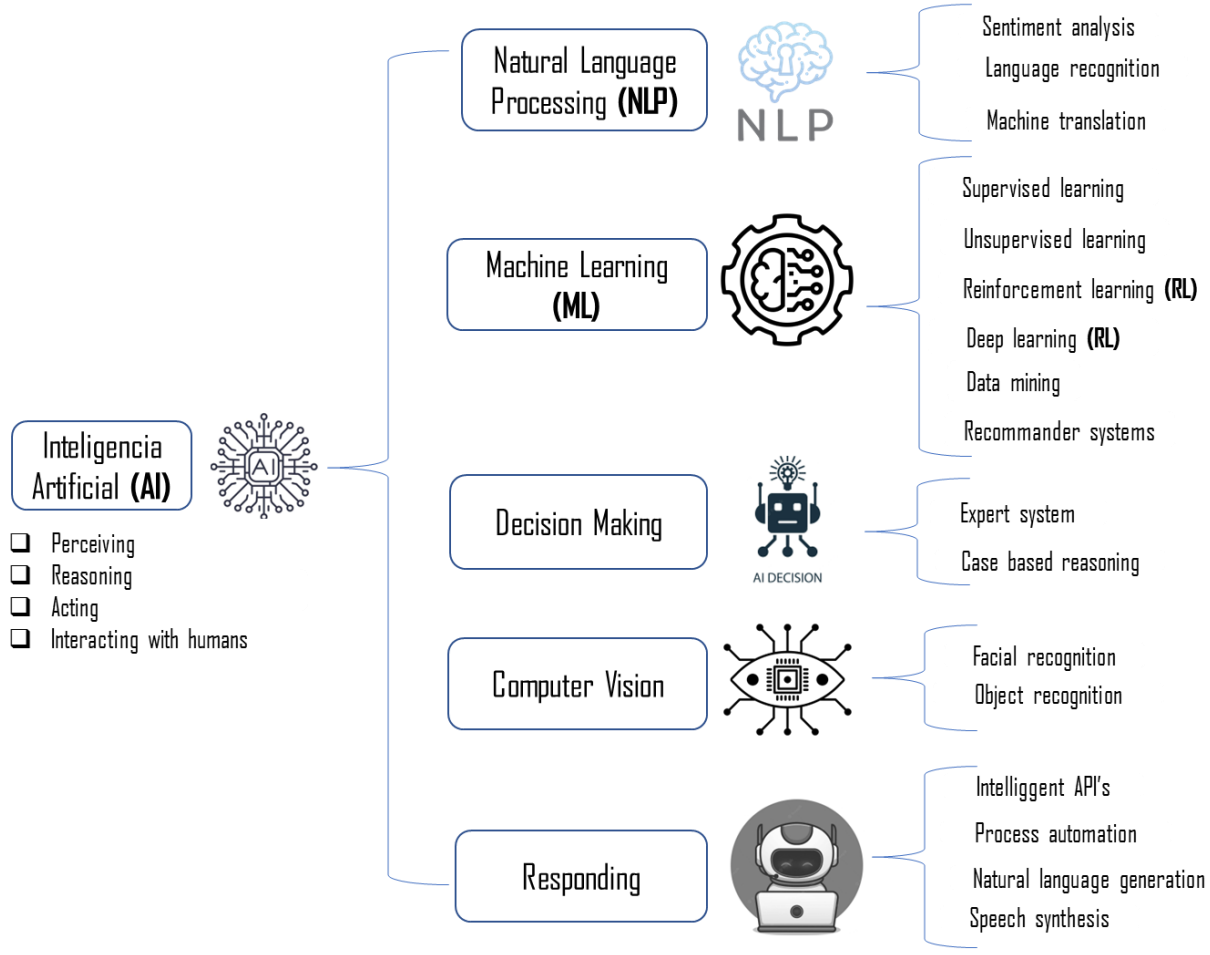
2.1. Inteligencia artificial (IA)

La inteligencia artificial es un campo de las ciencias de la computación que, en sinergia con otras tecnologías, busca que distintos dispositivos tecnológicos realicen tareas que normalmente requerirían inteligencia humana para llevarse a cabo, tratando de que estas emulen el comportamiento humano inteligente. La IA se puede aplicar en una amplia variedad de situaciones, como en gran parte de los ejemplos de la Figura 1, y es que debido a los beneficios que aporta, la IA se ha convertido un pilar fundamental para el desarrollo de la actividad industrial [6].

Sin embargo, replicar de manera fiable la capacidad y el comportamiento del cerebro humano a la hora de gestionar grandes conjuntos de datos para tomar una decisión o resolver un problema no es una

tarea simple. Es por ello que existen diferentes subcategorías con campos de estudio específicos dentro del campo de la IA, que surgieron como respuesta a esta necesidad, y es que cada uno de estos subcampos tienen modelos y procesos de aprendizaje distintos para adaptarse lo mejor posible a cada uno de los distintos usos que se le puede dar a la IA. A su vez, como se analizará más adelantes, dentro de estos propios subcampos, existen más clasificaciones, consiguiendo que la IA sea cada vez más versátil y pueda abarcar más ámbitos y dar respuesta a los problemas y desafíos que presenta la sociedad actual.

Figura 4 Clasificación de los campos de la Inteligencia Artificial.



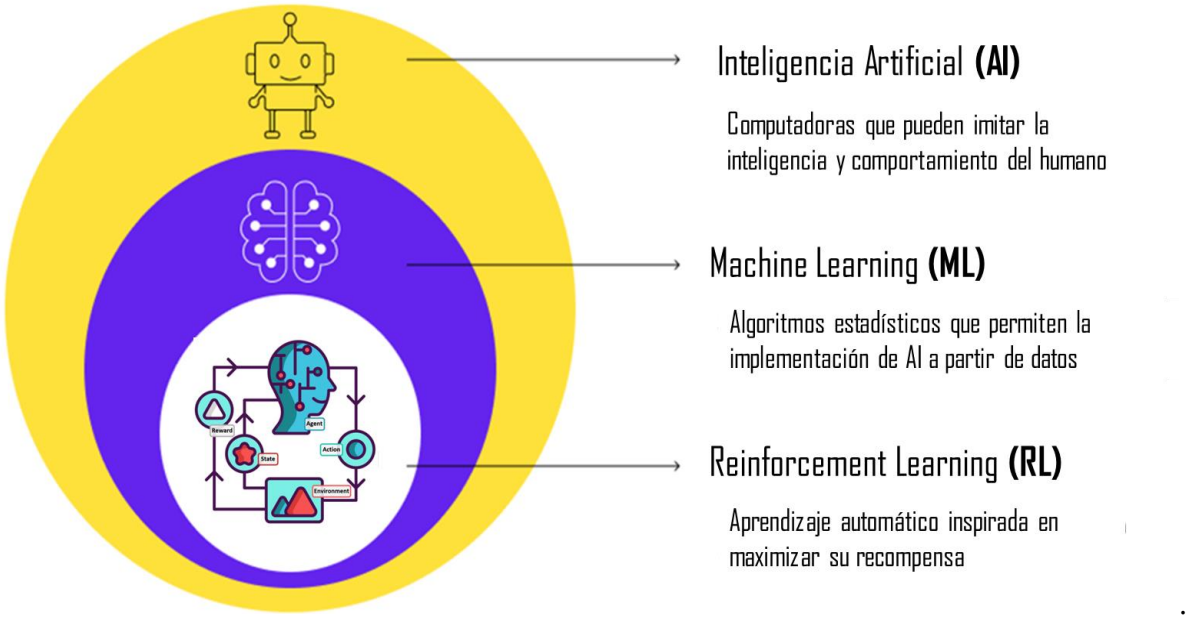
.

En la Figura 4 se puede apreciar los subcampos más destacados que de la IA, entre los que se encuentra Natural Language Processing (NLP), que se centra en dotar de capacidad a las máquinas para entender textos y palabras de la misma forma que lo hacen los seres humanos, Computer Vision, con el que los ordenadores pueden obtener información a partir de imágenes o videos como lo haría una persona, Decision Making, que una vez interpretados los datos, este se encarga de procesarlos y realizar predicciones precisas eligiendo la mejor acción para poder lograr su objetivo, Responding, que es capaza de generar respuestas o redactar como lo haría un humano y ML, que permite a los dispositivos tener un aprendizaje automático. A su vez dentro de cada una de estas clasificaciones se pueden apreciar los subcampos de cada una contiene [7].

Para el desarrollo de este trabajo fin de grado, en el que se trabaja con robots que deben desplazarse de forma autónoma para alcanzar las metas propuestas, ha sido necesario la implementación del subcampo de ML, debido a la capacidad de este para aprender conceptos de manera autónoma y llevarlos a la práctica, consiguiendo un comportamiento que lo hace único y destaca respecto al resto

[8]. A su vez, dentro del subcampo de ML, el tipo de IA que se ha desarrollado es el de RL, debido a que es el tipo que se adapta mejor a las necesidades de este trabajo, la relación entre los tres conceptos se puede apreciar en la Figura 5.

Figura 5 Correlación entre inteligencia artificial, aprendizaje automático y aprendizaje de refuerzo.



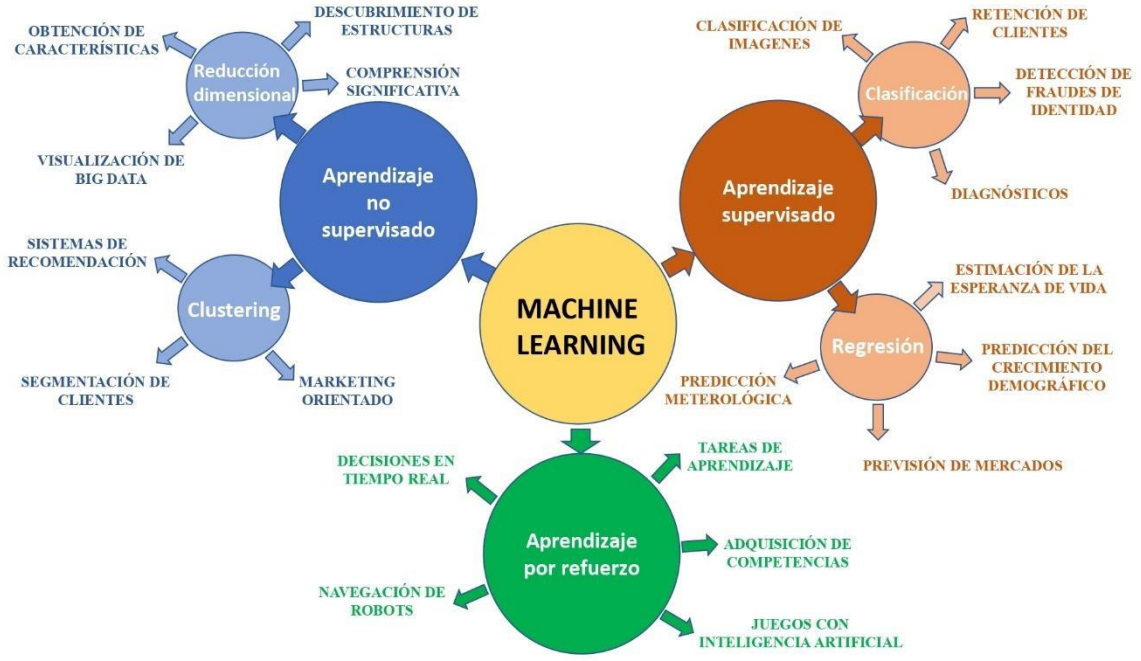
2.2. Machine Learning

Machine Learning es un campo de estudio que proporciona a los dispositivos capacidades de aprendizaje automático, con la habilidad de aprender una tarea sin estar explícitamente programados para realizarla y tomar decisiones óptimas similares a las que tomaría un humano.

El factor clave para que el aprendizaje automático sea exitoso y que con su uso se logre los objetivos determinados, es la capacidad que posee de obtener modelos complejos y aprender mediante una gran cantidad de datos de entrada, y es que esta es una de las principales características del ML y de la IA en su conjunto, y es la capacidad que tienen para manejar y trabajar con cantidades masivas de datos provenientes de distintas fuentes, necesitando por ello una capacidad y potencia computacional muy elevada para poder analizar correctamente todos los datos en un tiempo reducido [9].

En relación con los datos, su naturaleza y la dificultad a la hora de clasificarlos, ML se divide en tres métodos o subclases principales: Aprendizaje supervisado, no supervisado y aprendizaje reforzado [10]. Esta clasificación de los diferentes tipos de ML puede contradecir a la Figura 4, en la que se aprecia como ML se divide en 6 ramas, entre las que se encuentran las tres ahora comentadas junto *con Deep Learning, data mining y recommander systems*, esta clasificación también es válida, aunque realmente estas tres últimas se pueden considerar como subclases dentro de las tres primeras, ya que por ejemplo, Deep Learning puede ser tanto aprendizaje supervisado como no supervisado [11]. De esta forma, para clasificar los distintos tipos de ML lo más común es diferenciar entre las distintas técnicas de aprendizaje, siendo estos los ya comentado; aprendizaje supervisado, no supervisado y reforzado [7]. En la Figura 6 se aprecian los tres distintos campos y las principales aplicaciones de cada uno de ellos en la actualidad.

Figura 6 Campos del Machine Learning y sus principales aplicaciones. Fuente: elaboración propia a partir de [13].



En cuanto a las diferencias de estos tres tipos, en el primero, los datos utilizados para el entrenamiento han sido etiquetados u organizados previamente, creando de esta forma un sistema basado en un conjunto de muestras emparejadas de entrada-salida que condicionan el determinan el entrenamiento y el funcionamiento del sistema o algoritmo [14]. Por su parte, en el aprendizaje no supervisado, los datos que se utilizan para el entrenamiento no han sido etiquetados ni organizados ya que no han sido procesados previamente, por lo que este es un tipo de aprendizaje automático en el que se buscan patrones que no han sido detectados previamente en los datos al no estar etiquetados, a diferencia de lo que sucedía con el aprendizaje supervisado.

También es importante destacar un tipo de ML que se ha desarrollado a partir de estos dos últimos, que se muestra como un enfoque híbrido llamado aprendizaje semisupervisado, el cual dispone de tanto de datos ya previamente etiquetados como datos que no lo están, predominando el uso de estos últimos, lo que reduce el costo del entrenamiento [12].

El último tipo de ML es el aprendizaje reforzado, que como se explicará a continuación, basa su funcionamiento en una serie de recompensas o castigos que determinarán si el robot está tomando las decisiones adecuadas. En el siguiente apartado se explicará más en detalle este tipo de ML, ya que como se ha comentado antes, es el tipo de IA con el que se ha implementado todo el trabajo final de grado.

2.3. Reinforcement Learning

El aprendizaje reforzado (RL) busca dar cabida a situaciones o problemas que no se podrían resolver o que serían muy complicados y extensos de ejecutar con el aprendizaje no supervisado y principalmente en el caso del supervisado. Este es debido a que en algunos de los usos en los que se aplica ML, existen gran cantidad de datos y de variables, por lo que es posible que no todas estas puedan haber sido ejemplos directos del entrenamiento realizado, por lo que es necesario un método que abarque situaciones más complejas y extensas [15].

Un ejemplo de estas situaciones es la ya comentada conducción autónoma de coches o robots, en la que pueden influir infinidad de variables que hacen necesario el uso de RL, ya que, por ejemplo, utilizando aprendizaje reforzado serían necesario tener en cuenta gran parte de esas variables y

situaciones para el entrenamiento, lo que sería realmente complicado y costoso, llegando a ser imposible en muchos casos.

Por otra parte, el RL, al igual que en el aprendizaje no supervisado, no dispone de datos previamente etiquetados, pero se diferencia de este en que el RL busca maximizar las recompensas que obtiene por sus acciones, mientras que el aprendizaje no supervisado pretende encontrar una estructura oculta a partir de los datos no etiquetados [16].

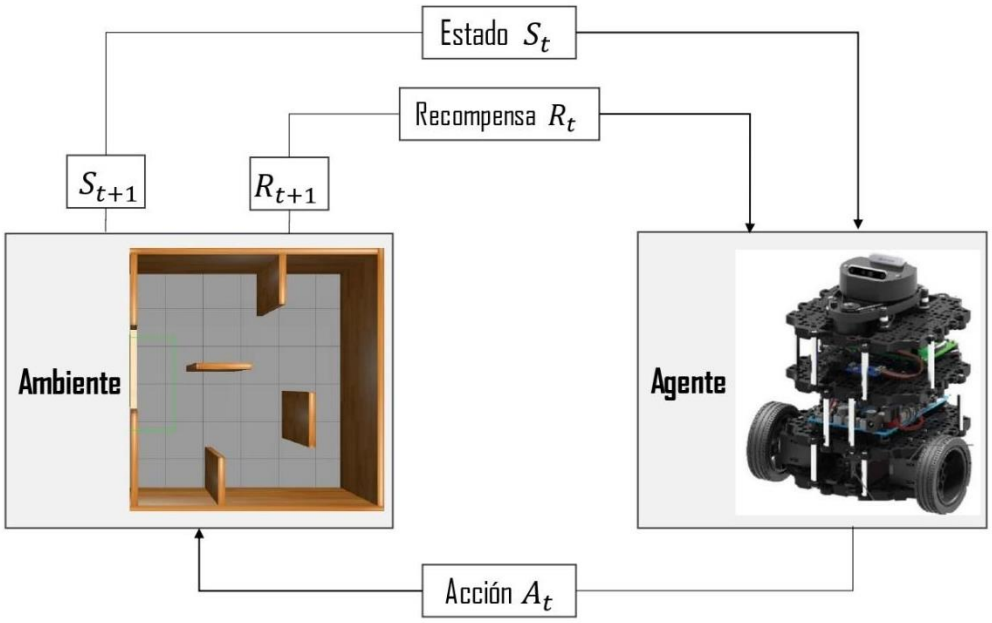
En cuanto al funcionamiento del RL, la clave para su correcto desarrollo y actuación es la interacción del agente con el entorno y de cómo aprende de él, ya que mediante RL se pretende mejorar esta interacción en función de la experiencia previa que se tiene del entorno, de la misma forma que lo realizan los seres humanos, para los que la experiencia es vital a la hora de tomar decisiones [17].

Para conseguir esto, el RL utiliza un sistema de recompensas con el que el agente, a partir de cada acción que ejecuta, obtiene una recompensa o un castigo, en función de si la acción ha sido buena o beneficiosa de cara al objetivo a alcanzar, o si, por el contrario, ha sido una acción errónea o no óptima que atrasa la consecución de este [17].

Otra característica propia del RL es el equilibrio en el uso de la exploración y la explotación, que se da debido a que, a diferencia del aprendizaje supervisado y no supervisado, en RL se tiene conocimientos previos sobre el entorno. La exploración se da cuando el agente no tiene conocimiento del entorno y busca objetivos desconocidos en él, para probar acciones que no ha probado antes, estando de esta manera explorando el entorno. Por su parte, la explotación se da cuando el agente ya tiene algún conocimiento del entorno, por lo que el agente explota lo conocido para obtener una buena recompensa al ya tener conocimientos de donde está [16].

De esta manera el agente debe encontrar un equilibrio entre ambas ya que la combinación de ambas da mejores resultadas que cada una por separado, debido a que, para obtener gran cantidad de recompensas tiene que explotar las acciones que ha ya ha realizado y que le garantizan buena recompensa, pero también debe explorar el entorno para descubrir nuevas acciones, ya que de esta manera conoce y aprende más sobre el entorno y puede conseguir mejores recompensas [16].

Figura 7 Funcionamiento de un algoritmo de Reinforcement Learning.



Con todo esto, en la Figura 7 se muestra el funcionamiento general de un algoritmo de RL, siendo los principales factores que influyen en el aprendizaje reforzado los siguientes:

- Agente: Aquel que se encuentra en el entorno y debe que tomar las decisiones adecuadas para alcanzar el objetivo propuesto.
- Entorno: Ambiente en donde se encuentra el agente, y a partir del cual se obtendrán los datos necesarios para tomar cada una de las decisiones.
- Estado (S): Son los datos que toma el agente del entorno, estos pueden ser muy variados, un ejemplo común serían los datos de distancia del robot frente a los obstáculos que tiene alrededor. Estos se suelen medir con sensores como un láser o una cámara.
- Acción (A): es el movimiento o decisión que realiza el agente en función del estado que ha obtenido del entorno.
- Recompensa (R): Es valor de castigo o recompensa que obtiene el agente una vez ha realizado la acción, y si esta ha sido óptima o no. A partir de esta recompensa, se desarrolla la política de comportamiento del agente.

De esta forma el funcionamiento en su conjunto sería: el agente obtiene el estado S_t partir de su situación y posición en el entorno en el momento actual t, después calcula y realiza la acción A_t en función del estado que ha obtenido y pasa a estar entonces en una nueva posición S_{t+1} y finalmente, se obtiene la recompensa R_{t+1} dependiendo de cómo de buena ha sido la acción con la que se sigue desarrollando la política de comportamiento del agente [18].

Con relación a esta última parte, el algoritmo de RL utilizado en este trabajo de fin de grado para modelizar la política del agente está basado a partir de Q-Learning (Q-L). El algoritmo se explica más en detalle en el siguiente apartado.

2.3.1. Double Deep Q-learning (DDQL)

Con la aparición del RL, muchos fueron los problemas que se vieron resueltos gracias a su desarrollo y aplicación en diferentes ámbitos, pero como sucede comúnmente en el mundo tecnológico, siempre se busca extender los campos de aplicación y cada vez las situaciones a resolver se vuelven más complejas. Es por ello, que, para solucionar problemas de alta dimensión, que eran inabarcables mediante algoritmos tradicionales de RL, se optó por incorporar el Deep Learning al RL, desarrollando el Deep Reinforcement Learning (DRL), dando solución a aquellos problemas de dimensiones elevadas, a partir del entrenamiento de redes neuronales profundas, consiguiendo funciones de valores y políticas de comportamiento mucho más óptimas [18].

De esta manera, se aplicaron las redes neuronales profundas en los distintos algoritmos de RL, uno de ellos el comentado Q-Learning, que es la el algoritmo base del que se utiliza para este trabajo de fin de grado, en el cual se aplicó Deep Learning consiguiendo el Deep Q-Learning (DQL), que combinaba Redes Neuronales Convolucionales (CNN en inglés) con Q-L. Finalmente, con el fin de mejorar las prestaciones de este último algoritmo, se desarrolló el DDQL, el cual mejoraba el rendimiento debido a que conseguía evitar las sobreestimaciones de las recompensas que se producían con DQL [19].

Para entender bien el funcionamiento del DDQL, que es el algoritmo utilizado en el trabajo, es importante conocer sus dos predecesores, ya que estos constituyen la base principal del algoritmo. El algoritmo Q-L basa en un método fuera de la política que diferencia entre la política de actuación y la política de aprendizaje [19], con esto se pretende desarrollar y aprender un conjunto de normas que le indiquen al agente la acción que debe ejecutar dependiendo de las circunstancias en las que se encuentre. La fórmula matemática que modela el Q-L en la que se busca el cálculo del valor Q, es la siguiente:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R + \gamma_{MAX}(S_{t+1}, a) - Q(S_t, A_t)] \quad (1)$$

- γ: Factor de descuento comprendido entre 0 y 1
- α: tasa de aprendizaje con la que el modelo aprende
- a: acción con más valor posible
- R: tasa de reducción de la recompensa conforme pasa el tiempo.
- $Q(S_t, A_t)$: valor esperado de la recompensa para una acción dada en el estado actual t
- $max(S_{t+1}, a)$: valor de la recompensa esperado de la recompensa en un estado futuro $t+1$ si la acción con más valor es llevada a cabo [20]

De esta forma, en el algoritmo Q-L, se actualiza el valor Q en cada estado utilizando la ecuación anterior, generando la llamada Q-tabla y actualizándola en cada iteración. Esta consiste en una tabla básica en la que se calcula las recompensas máximas esperadas para cada par de estado y acción [20], de esta manera en la Q-tabla se aprecia fácilmente de forma visual las recompensas que se obtendrían en cada estado si se realiza una acción u otra. En la Figura 8 se puede apreciar cuál sería la forma de una Q-tabla genérica:

Figura 8 Q-tabla genérica de un algoritmo Q-L a partir del estado y las distintas acciones posibles.

Acción

Estado	A_1	A_2	....	A_M
S_1	$Q(S_1, A_1)$:	$Q(S_1, A_2)$		$Q(S_1, A_M)$
S_2	$Q(S_2, A_1)$	$Q(S_2, A_2)$		$Q(S_2, A_M)$
⋮			⋱	
S_N	$Q(S_N, A_1)$	$Q(S_N, A_2)$	....	$Q(S_N, A_M)$

Con este método se consiguen, en entornos poco complejos con un solo agente, unos resultados en el aprendizaje realmente buenos. En cambio, en entornos muy grandes con gran cantidad de estados y acciones posibles, o con varios agentes involucrados, el rendimiento de este algoritmo disminuye considerablemente debido a que las Q-tablas se volverían demasiado grandes para trabajar con ellas al haber tantas posibilidades, siendo necesario mejorar el algoritmo, motivo por el cual se desarrolló el DQL.

Como se ha comentado al principio de este apartado, el DQL es una combinación del algoritmo ya explicado de Q-L junto con redes neuronales profundas, consiguiendo un algoritmo mucho más potente que puede ser usado para situaciones más complejas. Para conseguir esto se utiliza el mecanismo de *experience replay* conocido en español como repetición de la experiencia, con el que se pretende aleatorizar los datos para evitar las correlaciones en las observaciones, que es uno de los principales problemas que generan los algoritmos de RL, que tienden a ser inestables debido a casusas como la correlación [19].

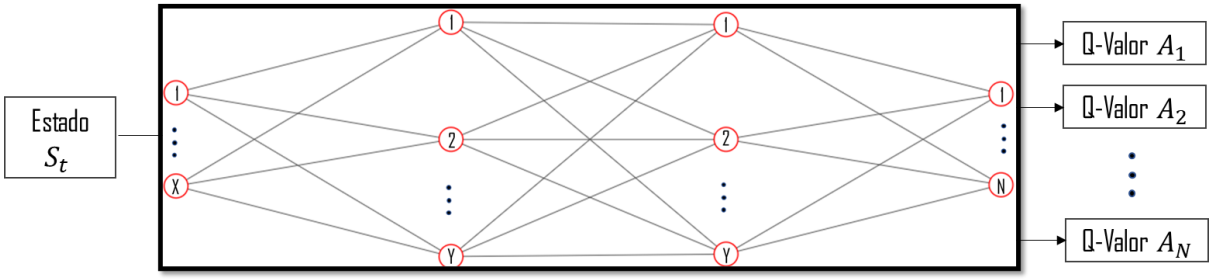
De esta manera, el mecanismo de repetición de la experiencia se basa en almacenar las últimas experiencias del agente en memorias de un tamaño fijo, para de esta manera poder utilizar estas muestras varias veces en el proceso de entrenamiento, consiguiendo así mejorar tanto la estabilidad como la eficiencia de la red. Los datos de experiencia que se almacenan en cada instante t están formados por la ecuación 2 mientras que las memorias formadas por las experiencias corresponden con la ecuación 3 [21]:

$$e_t = (s_t, a_t, r_t, s_{t+1}) \quad (2)$$

$$D_t = \{e_1, \dots, e_t\} \quad (3)$$

Como se aprecia en la primera ecuación, en el instante t se almacena el estado, la acción y la recompensa junto que el siguiente estado $t+1$. Mientras que, en el caso de las memorias, se almacenan las experiencias pasadas del agente. Con todo esto, la estructura del DQL ya no sería en forma de Q-tabla como en el caso de Q-L, sino que ahora ya estaría en función de la arquitectura de la red neuronal profunda, como se puede apreciar en la Figura 9:

Figura 9 Arquitectura de una DQL genérica.



Por último, como ya se ha comentado anteriormente, la siguiente evolución al DQL fue el DDQL que nació con el objetivo principal de suavizar o solucionar los problemas que presentaba el algoritmo DQL, principalmente en lo referido a las predicciones demasiado optimistas que se realizan con este. Además, con el DDQL se consiguió acelerar el tiempo de convergencia de la red, consiguiendo obtener un mejor rendimiento.

Para desarrollar este algoritmo se utilizan dos redes neuronales distintas que tienen exactamente la misma arquitectura, realizando cada una de ellas una tarea distinta. La primera red neuronal actúa como la red principal estando encargada de elegir la acción a realizar por el agente, es decir, calcula el valor Q, mientras que la segunda red se encarga de calcular el valor Q objetivo de la acción realizada. A la primera se le conoce como red Q (Q-network) y es entrenada en cada instante, mientras que la segunda, la red objetivo (target network) sólo se entrena cada cierto tiempo estando actualizada con los pesos de la red Q, de esta manera consiguiendo una política más estable al no estar actualizándose en cada instante [22]. La arquitectura de las redes neuronales utilizadas se explicará más adelante.

2.4. Pathplanning

El *pathplanning* se encarga de realizar deliberadamente la planificación local y global de la ruta que debe seguir el agente para desplazarse de un punto A hasta un punto B, a partir de construir y llevar a cabo una trayectoria que sea capaz de evadir aquellos obstáculos que el agente va registrando en el mapa del entorno que genera a partir de los datos extraídos del sensor a lo largo de su recorrido. El agente debe ser capaz de llevar a cabo este proceso y poder evadir obstáculos debido a que en el mundo real la mayoría de los entornos son complejos y variables.

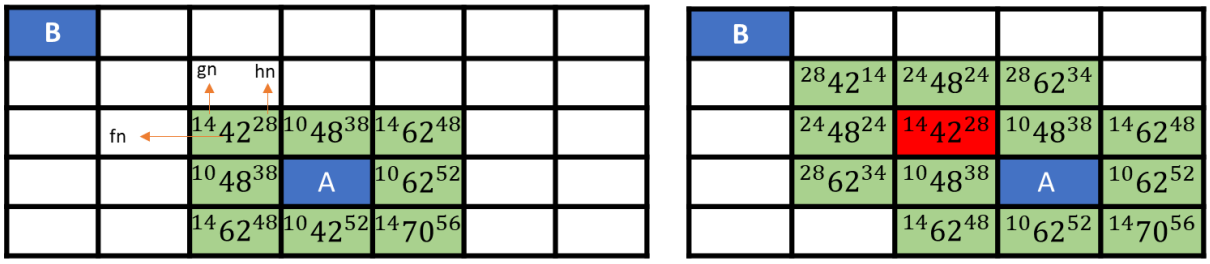
En la actualidad existen una gran variedad de algoritmos para encontrar el camino entre dos puntos, utilizando para este caso el algoritmo genético A*, el cual implementa una búsqueda heurística para expandir el camino potencial, pasando por diversas sub-metas hasta llegar a la meta, permitiendo al agente evadir obstáculos dinámicos que no se encuentren en el mapa generado, pero alcanzan a ser detectados por su sensor.

A* presenta un alto rendimiento, debido a que este entregara la ruta más corta, es decir, con el menor número de conexiones dentro de un conjunto de datos de la memoria de entrenamiento y las almacena para posteriormente llevarlas a cabo. Para poder encontrar la ruta con menor coste, el algoritmo divide el entorno en cuadriculas, las cuales son evaluadas y se le asigna a cada nodo un peso f_n según la ecuación 4 [23].

$$f_n = g_n + h_n \qquad (4)$$

Donde h_n es el costo en longitud de ir del nodo actual al nodo vecino y g_n representa el costo de la longitud de la distancia de ir desde el nodo de origen hasta el objetivo (meta). Y a partir de los pesos f_n, se selecciona el que menor peso presente computándola como una ruta valida y potencialmente optima. Este proceso se repite desde la situación inicial en la que se encuentre hasta llegar a su respectivo objetivo, logrando encontrar la ruta con el menor sumatorio de pesos f_n posible, este proceso se puede ejemplificar gráficamente con la Figura 10, en la que se muestran dos pasos realizados con el algoritmo A*.

Figura 10 Mapa de coste algoritmo A. Paso 1 y paso 2.*



Capítulo 3. Tecnologías utilizadas

Para llevar a cabo la realización de este trabajo de fin de grado se utilizará principalmente el lenguaje de programación Python, junto un entorno de trabajo especializado en simulaciones robóticas, así como diversas librerías específicas para el desarrollo de proyectos que impliquen el uso de robots e IA, siendo las principales tecnologías utilizadas las que se pueden observar en la Figura 11.

Figura 11 Logos de las principales tecnologías utilizadas en este trabajo de fin de grado.

3.1. Hardware y sistema operativo

El hardware utilizado para llevar a cabo este trabajo ha sido un ordenador de escritorio con una unidad central de procesamiento (CPU) Intel(R) Core(TM) i7 CPU 930 @ 2.80GHz formado por 8 núcleos, siendo el CPU el encargado de realizar el proceso de entrenamiento de las redes neuronales y no la unidad de procesamiento gráfico (GPU). Por su parte, el sistema operativo utilizado ha sido Ubuntu con la versión 16.04.

El software libre Ubuntu es una distribución de Linux, caracterizada por ser de código abierto, gratuito y con una gran comunidad de soporte en internet. El motivo por el cual se ha desarrollado este trabajo de fin de grado con este sistema operativo es debido al uso del entono Robot Operating System (ROS), que se explicará más adelante, ya que ROS está enfocado para su uso en sistemas operativos de la familia UNIX, como es el caso de Ubuntu.

3.2. Python

Se utilizará Python como lenguaje de programación principal, debido a que es uno de los principales lenguajes que existen en la actualidad para desarrollar algoritmos de IA y ML en concreto, gracias a los múltiples beneficios que ofrece para desarrollar este tipo de algoritmos. Python nació a finales de los años 80 de la mano del programador Guido van Rossum estando fundamentado en el lenguaje ABC. Desde su creación hasta la actualidad, son diferentes las versiones que se han desarrollado, siendo

Python 3.0 la última de ellas, lanzada a finales de 2008. Python se caracteriza por ser un lenguaje de programación interpretado, de alto nivel, orientado a objetos y multiplataforma [24].

El éxito de este lenguaje principalmente reside en lo accesible que son los códigos de Python, en el sentido que de que no resultan muy complejos a la hora de entenderlos o de escribirlos, y es que, desde su creación, los desarrolladores de Python siempre han defendido que parte de los principios del diseño debían ser que fuese un lenguaje explícito, simple, legible práctico y hermoso. Además, otro de los puntos fuertes de Python es la librería estándar que tiene, teniendo multitud de módulos y tópicos que ayudan al programador a la hora de desarrollar software, siendo Python uno de los lenguajes que en la hoy en día cuentan con una librería tan completa [24].

En cuanto al éxito del uso de Python para el desarrollo de algoritmos de ML, se debe principalmente a las características del lenguaje que se acaban de explicar y también a la disponibilidad de múltiples librerías y entornos de trabajo (frameworks) que facilitan en gran manera el trabajo, por lo que, de esta manera, se presenta como un lenguaje idóneo para desarrollar algoritmos de IA y ML. En relación con esto último, en Python existe tanto la librería NumPy y así como el entorno de trabajo TensorFlow, ambos utilizados en este trabajo fin de grado que se explicarán en la siguiente sección, que son muy importantes para desarrollar proyectos de aprendizaje profundo, entre otros [25].

A parte de las librerías y entonos de trabajo, los principios en los que se fundamenta Python son esenciales para que se presente como uno de los mejores lenguajes, o el mejor, para aplicaciones de ML. Esto es debido, en primer lugar, gracias a la legibilidad de los códigos, que supone un incentivo para nuevos desarrolladores aprender el lenguaje, además, el hecho de tener una sintaxis muy práctica, junto con las numerosas librerías de las que dispone, hace que el desarrollo de aplicaciones con Python sea más rápido comparado con otros lenguajes de programación más complejos. Por último, también existe en la actualidad una comunidad muy grande de Python que hace que haya mucho soporte y documentación online que ayuda a desarrolladores de todo el mundo a solucionar sus dudas y problemas [26].

3.3. TensorFlow

Entre las tecnologías comentadas anteriormente e ilustradas en la Figura 11 se encuentra la biblioteca de software TensorFlow, al ser con diferencia, la biblioteca de aprendizaje profundo más popular, con muchos modelos implementados y liberados, además de que dispone de diversas API en varios lenguajes, siendo la API de Karas de nuestro lenguaje principal Python la más completa y estable. TensorFlow es una herramienta flexible que permite a los usuarios programar y entrenar eficientemente una gran variedad de modelos de redes neuronales y otros modelos de aprendizaje profundo, para detectar patrones y razonamientos usados por los humanos y desplegarlos en producción [27].

Esta multiplataforma lleva a cabo operaciones matemáticas reales mediante diagrama de flujo de datos, que permite acelerar las tareas de aprendizaje en los pasos que realice del flujo de trabajo, logrando mantener un rendimiento óptimo y continúo reduciendo el tiempo de compilación. Adicionalmente, ofrece soporte de GPU, paralelismo de datos y la posibilidad de ser implementado de diferentes formas de manera local en una gran variedad de dispositivos, ya sea de integrada en una aplicación, en el navegador o en la nube.

3.4. Keras

Como se comentó con anterioridad, se utilizó una API relativamente simple, de alto nivel de redes neuronales escrita en Python construida sobre TensorFlow conocida como Keras. La cual es una biblioteca que itera a la velocidad del pensamiento que está diseñada para minimizar la carga cognitiva mediante la reducción de la cantidad de acciones de requeridas del usuario para casos de uso común.

Keras se encuentra estrechamente vinculado con TensorFlow y le brinda soporte para algunas funciones avanzadas de las fases del flujo de trabajo del aprendizaje, desde la capacidad de cargar los datos de manera eficiente, hasta el entrenamiento de los hiper-parámetros y las estrategias de aplicación. Keras y TensorFlow forman juntos un tándem de gran potencia y sencillez de uso que ofrece flexibilidad para la investigación, así como coherencia y rapidez para la implementación [28].

3.5. ROS

Uno de los pilares de este trabajo de fin de grado es la plataforma donde se ejecutan los robots, para ello se ha utilizado el entorno integrado de desarrollo conocido como ROS, el cual es un sistema operativo gratuito compuesto por un conjunto de bibliotecas de software y herramientas que facilita el desarrollo de software de plataformas robóticas. Así mismo, también se ha utilizado un entorno de simulación robótica 3D, conocido como Gazebo, con el fin de simular con precisión y eficiencia diferentes robots en varios escenarios para comprobar con el fin de comprobar cómo se adaptan a cada circunstancia. Así mismo también se han utilizado otras herramientas complementarias y librerías que facilitan la recolección, el tratamiento y la sincronización de la información y datos utilizados.

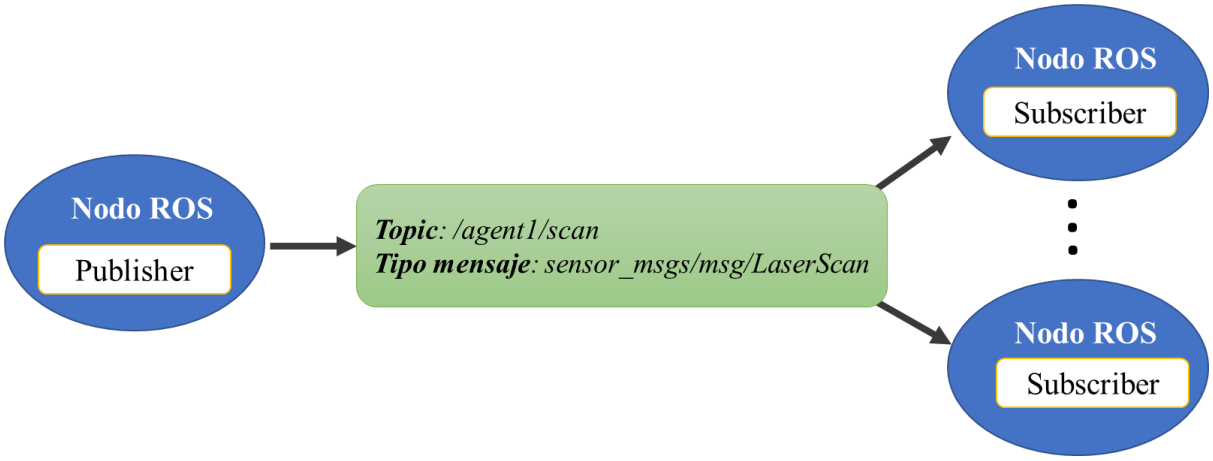
El motivo principal para utilizar ROS y Gazebo es que presentan una amplia variedad de potentes herramientas para trabajar al mismo tiempo que disponen de mucha información accesible en la red (librerías) de código abierto que facilita el desarrollo del proyecto. Como se ha comentado, uno de los primeros objetivos de este trabajo de fin de grado ha sido conocer en profundidad el funcionamiento de ROS debido a que es un pilar fundamental en el desarrollo del trabajo, así como por la importancia que tiene este software hoy en día en las aplicaciones de robótica.

Con el fin de conocer como se ha desarrollado el trabajo es necesario conocer las características y elementos principales de ROS para entender de qué forma se trabaja con este entorno de desarrollo. Entre los elementos de ROS destacan, los nodos, *topics* y mensajes:

- Nodos. Son procesos que realizan cálculos y funciones dentro del robot. Este normalmente está formado por diversos nodos en los que cada uno se encarga de diferentes acciones, como, por ejemplo, que se verá más adelante, el proceso de transformación y filtrado de datos del robot. Estos se escriben mediante librerías de cliente ROS, como son *roscpp* si es mediante el lenguaje C++ o *rospy* si es mediante Python, siendo esta la librería utilizada en este trabajo. Por último, los nodos se comunican entre sí mediante *topics* [29].
- *Topics.* Son buses de datos por los cuales los nodos se intercambian mensajes, siendo generalmente una comunicación y transmisión unidireccional. El sistema para este intercambio de mensajes se realiza a través de publicaciones y suscripciones, y cada *topic* solo utilizada un tipo de mensaje ROS [30].
- Mensajes ROS. Es la información que contienen los *topics* y presenta una estructura de datos simples en los que están implementados los distintos tipos de datos primitivos como son el booleano, número entero, de coma flotante, etc. Por su parte también pueden contienen vectores y matrices [31].
- *Publisher/Subscriber*. Hace referencia a que papel presenta un determinado nodo ROS. Como se ha comentado, los nodos intercambian mensajes de forma de unidireccional, de forma que hay un nodo que está obteniendo información del otro. El nodo que posee la información se le considera *Publisher* ya que está "publicando" lo que contiene, mientras que el *subscriber* es el nodo que se está "suscribiendo" al otro para obtener esa información.

En la Figura 12 se puede apreciar a modo de resumen el funcionamiento de la comunicación en ROS, en la que hay nodos que publican mientras que otros se suscriben a ese nodo. Esa comunicación se realiza mediante *topics* que transmiten un mensaje ROS que se caracteriza por ser de un tipo concreto.

Figura 12 Funcionamiento de la comunicación en ROS mediante topics entre Publisher y Subscriber.



3.6. Librerías

Para llevar a cabo de forma correcta y eficiente algoritmos complejos de IA, como lo son los basados en RL, es necesario utilizar librerías que aporten funcionalidades ya implementadas que faciliten el desarrollo del algoritmo. Entre las librerías de libre acceso más importante utilizadas en este trabajo destacan, Numpy, mpi4py y rospy.

3.6.1. Numpy

El nombre Numpy proviene de "Numeric Python" y es un paquete de computación implementado "open-source" esencial para realizar cálculos científicos mediante el lenguaje Python, debido a que permite al usuario aplicar de forma rápida, estable y optimizada algoritmos de aprendizaje, gracias a su capacidad de llevar a cabo funcionalidades y operaciones matemáticas de gran volumen de datos con estructura de dos o más dimensiones de forma veloz [32]. Las ventajas principales que ofrece Numpy son varias [33]:

- Es que es capaz de trabajar con matrices n-dimensionales de forma rápida y versátil, destacando la vectorización e indexación de estas, siendo Numpy un referente actual en relación con la computación de matrices.
- Numpy ofrece diversas herramientas de computación muy útiles como transformadas de Fourier o álgebra lineal, entre otros.
- Es completamente compatible con una gran variedad de hardware, bibliotecas y plataformas.
- El rendimiento es otro factor muy importante, y es que Numpy ofrece una velocidad de compilación muy elevada.
- Es una librería sencilla de usar debido a su sintaxis de alto nivel que hace que sea una accesible para programadores que están empezando.

3.6.2. MPI4Py.

Su nombre hace referencia, o significa, MPI para Python, y se ha convertido en el paquete o librería de Python más utilizada para desarrollar el estándar Message Passing Interface (MPI). Como bien su nombre indica, MPI es la interfaz de una biblioteca de envío de mensajes en paralelo, y se utiliza en procesos de paralelización, en los que se necesita el paso de mensaje entre múltiples procesadores. Por lo que con MPI se consigue enviar datos de la memoria de un proceso a la de otro debido a que cada unidad de procesador es independiente al resto con una memoria propia no compartida, por lo que a partir de las operaciones cooperativas que ofrece MPI se pueden comunicar y transmitir datos entre distintos procesadores, o núcleos [34].

MPI4py se presenta como un Application Programming Interface (API) orientada a objetos basada en MPI C/C++, que es en el lenguaje que está desarrollado el estándar. MPI4Py consigue mantener la semántica y sintaxis de MPI, permitiendo la comunicación paralela a través de buffers entre distintos procesos. La comunicación puede ser colectiva, de un núcleo a muchos, o punto a punto, pudiendo ser en ambos casos comunicaciones bloqueantes, en las que tanto el remisor o receptor de los datos no puedes realizar ninguna otra acción hasta que el proceso de comunicación haya finalizado, o la no bloqueante en la que el programa se continúa ejecutando mientras se están enviando o recibiendo los mensajes [35].

3.6.3. Rospy

Rospy es una biblioteca de cliente de Python específica para el entorno ROS. La principal función de la API de Rospy es proporcionar a los programadores de Python la capacidad de interactuar rápidamente con distintos procesos de ROS como son los servicios, parámetros y tópicos, siendo la rápida implementación la principal de sus ventajas [36]

En otras palabras, Rospy se puede considerar como un nexo de unión y de transmisión entre el código Python y ROS, ofreciendo servicios muy útiles para el programador, como por ejemplo la función *rospy.Subscriber* con la cual desde el código Python, el programador se subscribe a algunos de los *topics* que genera el robot una vez está ejecutándose en ROS, de manera que gracias a esa función de Rospy se puede recibir la información y los mensajes que está generando el robot.

Capítulo 4. Metodología

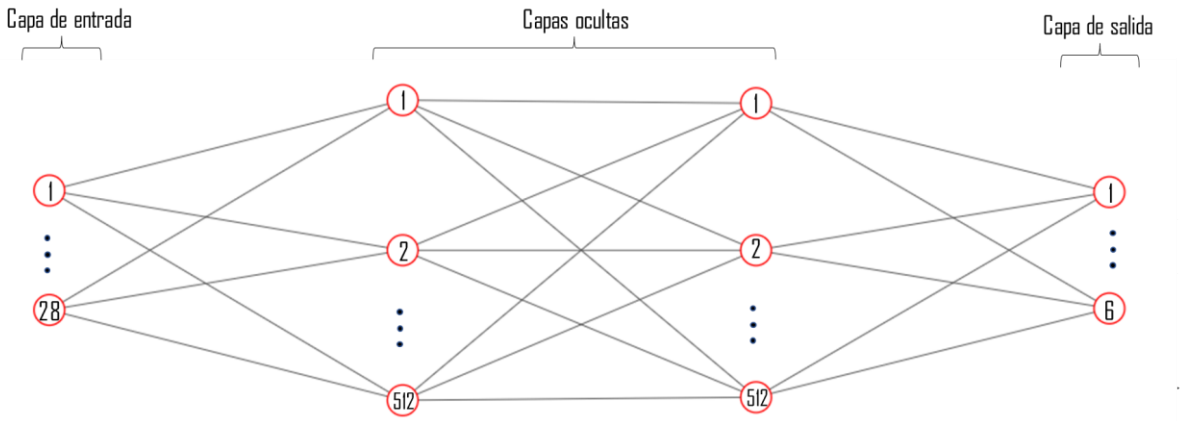
Este trabajo se centra en proporcionar una matriz de datos de entrada que cumpla con los requerimientos del algoritmo usado, utilizando sensores heterogéneos, de forma que aunque se utilicen sensores con menor precisión y ángulo de visión reducido, a través del procesamiento de los datos se consiga que el rendimiento sea similar y los robots con estos sensores puedan alcanzar las diversas metas establecidas de forma eficiente evitando colisionar con las paredes y obstáculos que van encontrando conforme se desplazan por el entorno.

Para conseguir la consecución de las metas, el proceso general que sigue el robot consiste en que este lee los datos del entorno a partir de los sensores que tiene y con estos datos se calcula la mejor acción a realizar por el robot para alcanzar la meta, repitiéndose este proceso hasta que el robot alcanza una política de comportamiento óptima, siendo este el funcionamiento general del RL, como se ha explicado anteriormente en el marco teórico y se aprecia en la Figura 7. Por otro lado, los datos recolectados son usados en paralelo para entrenar una política de comportamiento que será la responsable de decidir las acciones a ejecutar por el robot.

4.1. Red Neuronal

Como se ha comentado, el algoritmo utilizado consiste en dos redes neuronales idénticas, que tienen dos capas ocultas densas profundamente conectadas, llamadas así porque cada neurona recibe información de cada una de las neuronas de la capa anterior. Esta red neuronal puede apreciarse de forma gráfica en la Figura 13, que muestra la arquitectura de las redes donde se aprecia el tamaño de cada una de las capas:

Figura 13 Arquitectura de las dos redes neuronales con dos capas ocultas utilizadas en el trabajo.



- Capa de entrada: Array de 28 valores numéricos.
- Capas Ocultas: Dos capas densas de 512 neuronas cada una con función de activación Unidad Lineal Rectificada (ReLU).
- Capa de salida: 6 valores para cada una de las posibles acciones a realizar por el agente.

La función de activación ReLU es aquella que anula los valores de entrada negativos poniéndolos con valor 0, y no modifica los valores positivos, por lo que los valores serán o un valor positivo, o cero. En cuanto a la capa de entrada de la red neuronal, como se observa en la Figura 13, está compuesto por 28 valores de entrada, donde:

- 24 valores de la distancia entre el agente y su entorno, medidos a partir del sensor que tiene incorporado, bien láser con un láser o bien una cámara. Los valores medidos están equiespaciados alrededor del agente, es decir, los 24 datos representan los 360 grados del entorno de este.
- Ángulo de orientación actual del agente respecto al objetivo.
- Distancia actual entre el agente y el objetivo.
- Mínima distancia a la pared. Este es el valor mínimo de los 24 valores medidos por el sensor.
- Ángulo de orientación inicial del agente respecto al objetivo.

Estos son los datos que se obtienen del entorno del agente por los sensores en cada paso que realiza el agente, y que se utilizan para entrenar las redes neuronales y para calcular la acción a realizar en cada momento. Por lo tanto, el proceso que sigue el robot en cada paso que ejecuta se resume en: la recolección de datos, el proceso de entrenamiento de las redes, el cálculo de la acción a realizar y la ejecución de la acción por parte del robot. Estos procesos pueden ser llevados a cabo de diferentes formas, como, por ejemplo, que todos ellos los realice el propio agente, o que alguno de los procesos los realice un servidor externo. Para este trabajo, pensando en una posible implementación futura del proyecto, se ha buscado que no todos los procesos sean realizados por el agente.

Por este motivo, inicialmente se buscó que el agente fuese sencillo computacionalmente, y que se solo se encargara de llevar a cabo la recolección de los datos referentes al entorno en el que se encuentra y realizar la acción que corresponde en cada momento. Para conseguir esto, el agente después de cada lectura de los datos del entorno debe transmitirlos a un servidor, o "nube", encargado de llevar a cabo los procesos de entrenamiento de las redes neuronales y cálculo de la acción, enviando este servidor de vuelta al agente la acción a realizar. El motivo de externalizar los procesos de entrenamiento y cálculo de las acciones a un elemento externo es porque estos procesos son los que más carga computacional requieren, al ser los que implican el uso de las redes neuronales, por lo que el agente solo llevaría a cabo los procesos menos costosos.

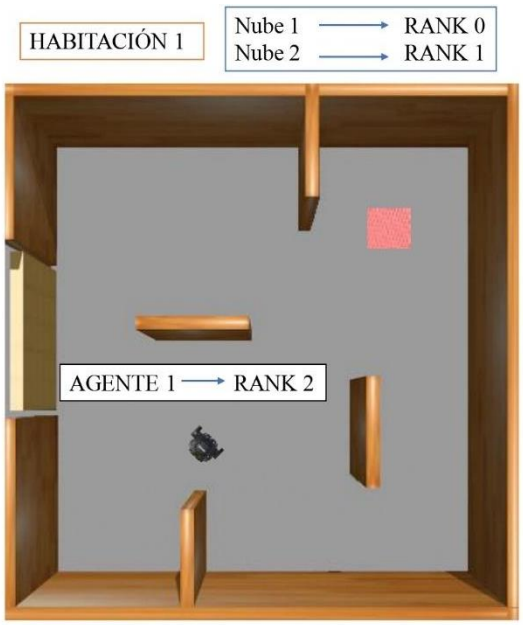
Pensando en llevar este proyecto a una implementación física, para conseguir que estos procesos separados entre agente y servidor se ejecuten de manera fiable y eficiente, sería fundamental el uso de la tecnología 5G o similar, que permitan transmitir gran cantidad de información con la mayor velocidad posible con una latencia mínima. No obstante, al no poseer una infraestructura con capacidad de red 5G que nos permitiese establecer la comunicación de los procesos, se optó por utilizar la técnica de paralelización con el fin de imitar el funcionamiento y lograr una comunicación de los procesos en tiempo real utilizando un solo CPU.

Gracias al *mpi4py*, aun encontrándose el agente y el servidor en un mismo ordenador, estos no se van a ejecutar de forma secuencial, sino que se separan en dos partes que se ejecutan de forma independiente y simultánea. La paralelización permite imitar el comportamiento de conexión de un ordenador con otro mediante el uso de los diferentes núcleos del CPU que presente, en donde se asigna uno o varios procesos a cada uno de los núcleos del CPU. Para este trabajo de fin de grado se asignó que un núcleo del CPU se encargue de ejecutar la parte del agente, y otro núcleo se encargue de la parte del servidor.

Sin embargo, a la hora de llevar a cabo el trabajo, fue necesario separar el servidor en dos partes, debido a que el núcleo encargado de realizar las tareas de este no era capaz de funcionar eficientemente debido a la carga computacional alta que requiere el entrenamiento de las dos redes neuronales y la utilización de una de las redes para calcular la acción a realizar. De esta forma, se optó por separar el servidor en dos; la nube 1, encargada del entrenamiento de las redes neuronales y la nube 2, encargada de la selección de la acción a partir de los datos proporcionados por el agente y de los pesos del entrenamiento realizado en la nube 1. Siendo necesario el uso de 3 núcleos distintos del CPU; uno para el agente, los otros dos para cada una de las nubes, para conseguir el funcionamiento adecuado de la paralelización.

En resumen, el reparto de núcleos se realiza según el número de agentes y ambientes, o habitaciones, existentes; donde cada agente siempre dispondrá de un núcleo exclusivo, por lo que, si hay 3 agentes distintos, habrá 3 núcleos dedicados para cada uno, mientras que cada habitación tiene 2 núcleos; uno para la nube 1 y el otra para la nube 2. En este trabajo, las pruebas se han realizado en una única habitación, por lo que ha sido necesario utilizar 3 núcleos (o *ranks*, que se les denomina a los diferentes núcleos en el paquete MPI4py), repartidos como se aprecian en la Figura 14:

Figura 14 Distribución de los núcleos, ranks, según el número de agentes y habitaciones.



Por otra parte, por ejemplo, si en la habitación hay dos robots, se necesitarán 4 núcleos, uno para cada robot, y otros 2 para las nubes de la habitación, y si, por el contrario, hay dos habitaciones y dos robots, uno entrenando en cada habitación, se necesitan 6; un núcleo para cada robot y dos para cada habitación.

Una vez explicada la asignación de los distintos núcleos del CPU en función del número de agentes y de habitaciones, se va a explicar a continuación las funciones propias que tiene cada una de las 3 partes implicadas; el agente, nube 1 y nube 2. Para ello es importante destacar que, de las dos nubes, la segunda, la que elije la acción, necesita llevar un registro de los agentes que hay dentro la habitación para poder diferenciar entre cada uno de ellos a la hora de recibir el estado de cada agente y enviar la acción a realizar que le corresponde a cada uno. Por su parte, la nube 1 no necesita llevar un registro de quien está en la habitación, debido a que ella únicamente entrena las redes neuronales con los datos que le envían, sin necesidad de saber si hay uno o más agentes en la habitación.

4.1.1. Agente

Como ya se ha comentado, el agente tiene la principal función de leer los datos del sensor, enviárselos a la nube 2 para que esta le indique la acción, y después realizarla. Además, tanto al principio debe indicar a la nube 2 de la habitación de que está en ella, y en caso de que hubiese otra habitación y se fuese a ella, avisa a esta de la nueva habitación de que ha entrado y también a la nube 2 de la antigua indicándole de que ha salido de ese espacio. Por último, el agente es encargado también de enviar a la nube 1 las memorias con las que esta realiza el entrenamiento de las dos redes neuronales. Por lo que, de forma sintetizada este es el proceso que sigue el agente en cada paso:

1. Envía el estado en el que se encuentra (los 28 valores de entrada de las redes neuronales) a la nube 2, junto con un parámetro, la variable *Pa*, necesaria para el cálculo de la acción, que se actualiza cada vez que el agente colisiona o pasa determinado número de pasos sin colisionar.

2. Recibe de la nube 2 la acción a realizar en función del estado que le ha enviado.
3. Ejecuta la acción que le ha enviado la nube 2 de la red neuronal, o si es el al principio del entrenamiento, ejecuta la acción que calcula el robot mediante las reglas.
4. Calcula la recompensa de la acción y lee el nuevo estado del robot.
5. Actualiza las memorias. Añade a la memoria existente los datos del estado, la acción que ha realizado, la recompensa, el estado siguiente que acaba de leer y una variable indicando si el agente ha colisionado o no.
6. Cuando las memorias alcanzan un tamaño determinado, son enviadas a la nube 1, y después son reiniciadas.
7. Comprueba si el agente se mantiene en la habitación o se ha desplazado a otra nueva.
8. En caso de estar en una nueva, el agente envía un mensaje a la nube 2 de la nueva habitación para indicarle que ha entrado y otro mensaje a la nube 2 de la antigua para que lo elimine de su registro.
9. Si ha alcanzado la meta, se calcula la recompensa total y se modifican las memorias.
10. Si el agente colisiona envía un aviso a la nube 1 para actualizar los parámetros del entrenamiento.

4.1.2. Nube 1

En cuanto a la primera de las nubes, esta es tiene la principal función de entrenar las dos redes neuronales a partir de las memorias que le envía el agente, y después enviar los pesos del entrenamiento a la nube 2 para que esta pueda calcular la acción que debe realizar el agente en cada momento. A continuación, se muestra lo que realiza la nube 1 en cada paso:

1. Entrena la red neuronal *Q-network* siempre y cuando las memorias tengan un tamaño adecuado.
2. Actualiza *target network.*
3. Guarda un modelo del entrenamiento.
4. Si el agente ha colisionado o ha pasado ciertos pasos sin colisionar, recibe un mensaje del agente para actualizar parámetros del entrenamiento. Es el mismo parámetro que envía el agente a la nube 2.
5. Si el agente ha enviado las memorias, la nube 1 las recibe y las almacena para después utilizarlas en el entrenamiento.
6. Obtiene los pesos de la *Q-network* y los envía a la nube 2, ya que esta es la red neuronal que se utiliza para calcular la acción.

4.1.3. Nube 2

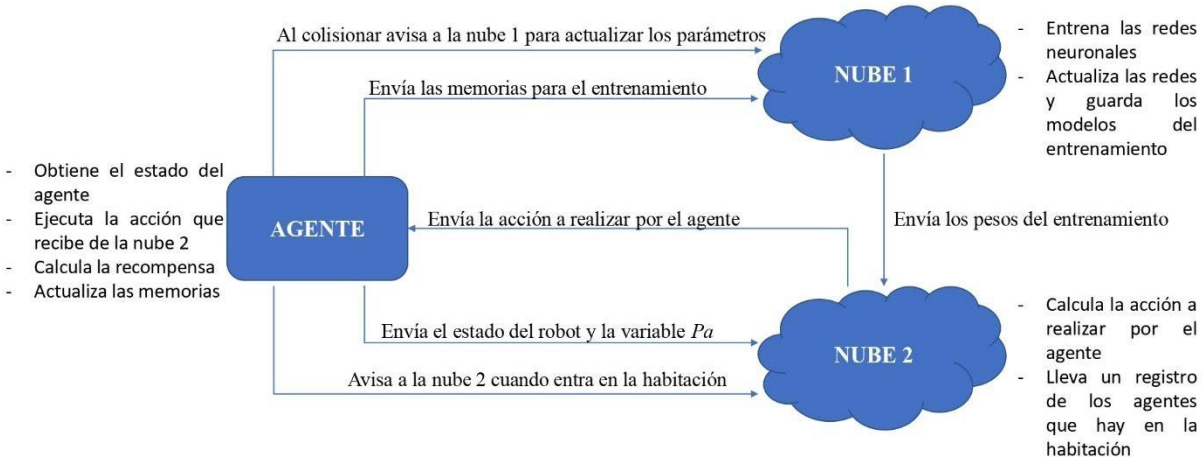
Por último, se encuentra la segunda nube, la cual tiene como función principal el calcular la acción a realizar por el agente, a partir del estado que le envía este y de los pesos que tiene de la red neuronal entrenada por la nube 1. Esta nube sí que diferencia si hay uno o varios robots en su habitación, por lo que en el proceso de recibir los estados de los agentes y de enviarles las acciones correspondientes, debe haber un control de los agentes para que los datos que se reciben y se envían tengan los agentes destinatarios correctos. Estos son las acciones que se realizan en cada uno de los pasos de la nube 2:

1. La primera vez que aparece el agente y cada vez que entra uno nuevo, recibe un mensaje de este y la nube 2 lo agrega en su registro de agentes existentes en la habitación.
2. Si algunos de los agentes que había en la habitación se ha ido a otra distinta, recibe un mensaje de este y la nube 2 lo elimina de su registro de agentes de la habitación.
3. Recibe el estado en el que se encuentra el agente y la variable *Pa* que utilizar para determinar la acción.

4. Calcula con la *Q-network*, actualizada con los pesos procedentes de la nube 1, la acción que debe realizar el agente.
5. Envía a este la acción para que la ejecute.
6. Si hay más de un agente en la habitación, se repiten los pasos 3, 4 y 5 para cada uno de los agentes.
7. Cuando la nube 1 los envía, recibe los pesos de la *Q-network*, para utilizarlos en el cálculo de las acciones.

En la Figura 15se puede apreciar de forma esquemática cada uno de los procesos que tiene que realizar cada parte y cuál es la comunicación que hay entre cada una.

Figura 15 Procesos a realizar por cada núcleo y comunicación entre ellos.



4.2. Agente utilizado

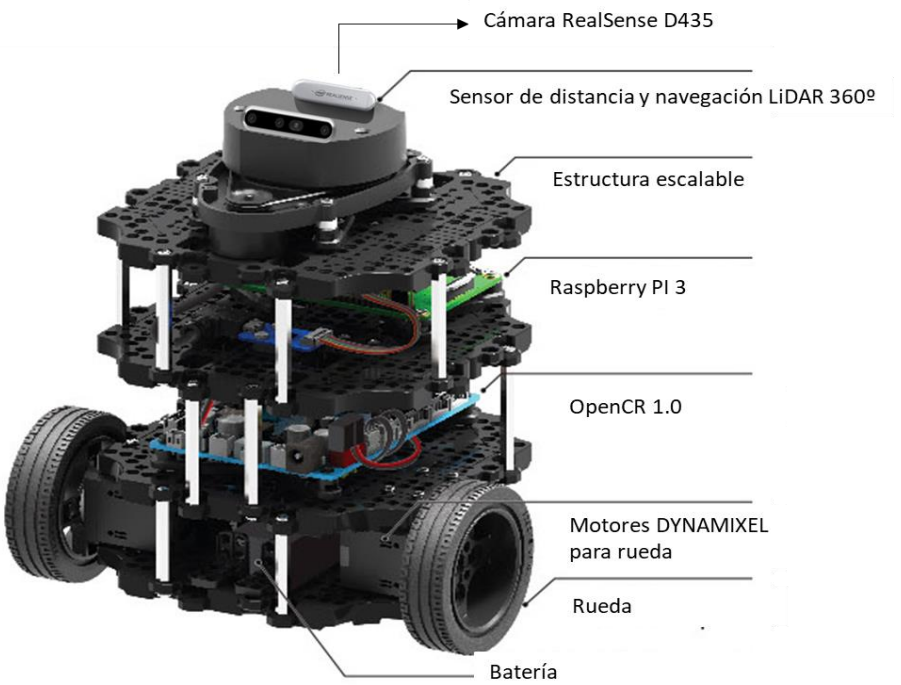
Como ya se ha comentado, con la aplicación del ML, y en concreto el RL para la conducción autónoma, se pretende que agentes, como pueden ser robots, traten de imitar el comportamiento que tendría el ser humano en una situación similar, por lo que cuanto mejor simulen ese comportamiento humano, mejor será la tarea que desarrolle el robot. De esta manera, el agente debe tener un comportamiento estable y continuo hacia la meta establecida, de forma que alcance los objetivos de forma óptima y en el menor tiempo posible.

El robot seleccionado para realizar este trabajo ha sido uno de la familia de TurtleBot, que ofrece robots personales caracterizados por tener software de código abierto y ser de bajo coste [37]. De las distintas generaciones disponibles de estos, se ha optado por TurtleBot3, que son robots personalizables, compactos, y programables, basados en ROS, teniendo un uso que se extiende desde la educación hasta la investigación. Se caracteriza por tener una tecnología Simultaneous Localization And Mapping (SLAM), que permite generar un mapa del entorno mientras el robot está navegando por él, además de que puede ser conducido de forma remota en caso de no querer que funcione de forma autónoma [38].

TurtleBot3 dispone de tres versiones; *waffle, waffle pi* y *burger*, que comporten gran parte de las características principales, con diferencias en la forma física que presenta cada robot y algún sensor concreto que llevan incorporado de serie. De las tres versiones, para este trabajo se ha utilizado TurtleBot3 burger, que es el más económico de los tres aun manteniendo las mismas buenas prestaciones en capacidad, funcionalidad y calidad [39]. Aunque para el trabajo no se ha utilizado un robot físico, sino en simulación, el aspecto económico siempre se ha mantenido presente, como parámetro a valorar por si se consiguiese llega a implementar el proyecto de forma física.

En la Figura 16, se puede apreciar el aspecto del TurtleBot3 Burger, con los distintos componentes que lleva incorporado, estando en la parte superior los dos sensores utilizados, láser y cámara, que se explicarán en las siguientes secciones. Es importante destacar que las cámaras incorporadas no venían de serie con el TurtleBot3 Burger, sino que se han tenido que ser instaladas e incorporadas en el robot mediante paquetes externos para poder llevar a cabo el trabajo, por lo que de todos los elementos que se aprecian en la Figura 16, todos veían de serie con el paquete TurtleBot3 Burger, a excepción de las ya comentadas cámaras.

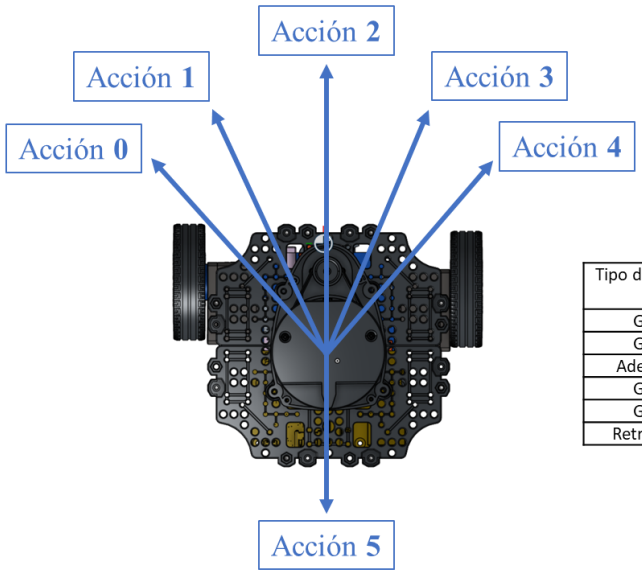
Figura 16 Estructura del robot de la serie Burger del TurtleBot3. Fuente: elaboración propia a partir de [40].



4.2.1. Movimiento del agente

Antes de explicar el funcionamiento del robot y los datos heterogéneos que proporcionan, es importante conocer cómo se mueve el robot, debido a que, tanto los posibles movimientos como la velocidad de cada uno de ellos, se determinan en el código desarrollado y serán importantes en el proceso de homogeneización de los datos. Como se ha visto anteriormente en la capa de salida de las redes neuronales, se ha determinado que el robot pueda hacer 6 acciones o movimientos distintos, cada uno con una velocidad lineal y angular distinta, como se puede apreciar en la Figura 17, en la que se aprecia la dirección de cada movimiento, así como la velocidad de cada uno:

Figura 17 Movimientos que puede realizar el agente junto con la velocidad lineal y angular de cada uno de ellos.



Tipo de acción	# acción	Velocidad lineal (m/s)	Velocidad angular (rad/s)
Giro	0	0.12	1.5
Giro	1	0.12	0.75
Adelante	2	0.15	0
Giro	3	0.12	-0.75
Giro	4	0.12	-1.5
Retroceso	5	-0.15	0

Es importante mencionar que dado a que nuestro trabajo se enfoca en la entrada y salida del algoritmo. Solo nos interesa las 5 acciones que permiten al robot avanzar hacia su objetivo y no la acción que hace que el robot retroceda, que solo se realiza al principio del proceso de entrenamiento del robot.

4.3. Datos heterogéneos

Los datos heterogéneos son la parte principal del trabajo y consisten en el tratamiento de los datos que lee el robot de su entorno, ya que son los datos con los que se realiza el entrenamiento de las dos redes neuronales y es a partir de ellos que se calcula la acción a realizar por el robot en cada instante. Por este motivo el manejo y tratamiento de los datos debe ser rápido, preciso y eficaz para que el robot funcione correctamente en todo momento. La recolección de estos datos se realiza a partir de sensores incorporados en el cuerpo del agente, y es que estos sensores juegan un papel crucial en los robots autónomos, ya que son la herramienta con la que los robots conocen lo que les rodea, y a su vez, permiten al programador, a partir de los sensores y sus datos, modelar el funcionamiento que tendrá el agente.

Los tipos de sensores que pueden llevar incorporados los robots son múltiples y muy variados, ya que, lo normal es que cada uno de ellos aporte información diferente que complementada hagan que el robot funcione correctamente. Alguno de los sensores más comunes son los de distancia, que indican la distancia lineal entre el agente y lo que tiene alrededor, los de posición, que permiten determinar al agente en que posición se encuentra en un espacio o respecto a otro punto, los sensores de imagen, como una cámara, que permite identificar al agente qué tiene a su alrededor, o aquellos sensores que se encargan de medir parámetros del ambiente como los sensores de humedad, de luz, de presión o de sonido, entre otros [41], [42].

De todos los sensores posibles, este trabajo de fin de grado se centra en aquellos que puedan determinar la distancia entre el agente y su entorno, por lo que se trabajará entonces con dos tipos de sensores para medir la distancia:

- Sensores de distancia mediante láseres, al ser el tipo de sensor más común y práctico para medir la distancia del robot con su entorno. Además, una de sus principales características es la precisión que tiene en comparación con otras formas de medición. Un aspecto negativo es que, debido a las elevadas prestaciones que ofrecen los láseres, suelen ser costosos económicamente.

- Cámaras profundas, que no son propiamente sensores de distancia, pero que se transformarán los datos obtenidos en ellas para convertirlos en datos de distancia. Se han seleccionado cámaras de profundidad, debido a que con ellas se permite hacer una estimación de la distancia a los objetos que son captados por ellas, resultando este tipo de sensor más económico que los láseres.

Es por esto que en este trabajo se habla de datos heterogéneos, debido a que se trabaja con dos tipos de datos completamente distintos, los procedentes del láser y los de la cámara, que no son de la misma naturaleza, pero que, con el tratamiento adecuado, se consiguen adecuar de forma que el robot, teniendo un tipo de sensor u otro, funciona de la misma manera. Esto es posible gracias a la conversión de los datos de la cámara en datos con la estructura de un sensor de distancia, como son los del láser, más adelante se desarrollará más en profundidad como se consigue esto.

Es importante destacar que la razón por el cual se necesita convertir los datos de la cámara al formato del láser es porque la estructura y el tamaño de los datos de entrada de las redes neuronales deben ser los siempre los mismo, independientemente del tipo de sensor con el que se lean los datos. Como se ha explica en la sección de las redes neuronales, son los 24 valores de distancia de los objetos de alrededor del robot, junto con otros 4 valores, los que forman los 28 valores que son los datos de entrada de las redes, siendo esos 24 valores de distancia, los que aporta directamente el láser. Por otra parte, como con la cámara no se obtienen los datos en este formato, es necesario hacer la transformación de sus datos para que la cámara también arroje 24 valores de distancia para ser utilizado en las redes neuronales.

En los siguientes apartados se explicarán brevemente las características de los sensores utilizados, así como los datos que se obtienen con cada uno de ellos.

4.3.1. Laser

El sensor ya adoptado e integrado de serie en el TurltleBot3 Burger es un sensor de rango de distancia laser (LDS-01) que utiliza la tecnología *Light Detection and Ranging* (LiDAR)**,** que consiste en transmitir constantemente pulsos de luz cortos que, al colisionar con un obstáculo, se reflejan y a partir del tiempo transcurrido entre emisión y recepción del pulso de luz, conociendo la velocidad de transmisión de esta, se puede calcular la distancia hasta el objeto con el que ha colisionado [43].

Figura 18 Sensor de rango de distancia laser LDS-01[40].

En la Figura 18 se puede apreciar la forma del láser, mientras que en la Figura 16 se puede ver a este montado en la parte superior del cabezal giratorio del robot. Este sensor es un escáner láser 2D con el cual se puede mapear y recopilar un conjunto de datos en un entorno circundante cubriendo los 360º del robot con gran precisión, con un alcance mínimo de 120 mm, un alcance máximo de 3.5 metros y una tasa de muestreo de 1.8kHz. De esta forma este sensor consigue desarrollar mapas tridimensionales precisos y de fácil procesamiento, que son realmente útiles para ser utilizados en la tecnología SLAM [40].

Como ya se ha comentado, el láser LDS-01 es el sensor de cálculo de distancia que ya venía integrado en el TurltleBot3 Burger, además de ser, de los sensores disponibles, a priori el más preciso en sus estimaciones, por lo que se decidió que los valores proporcionados por el láser serían los de referencia a la hora de establecer el formato de los datos de entrada de lase redes neuronales. De esta forma, se considera que el láser es el sensor de referencia de este trabajo, debido a que se da por supuesto que es el que funciona mejor, o el que va a proporcionar los datos más fiables para el proceso de entrenamiento de las redes. De esta forma, los datos del otro sensor, la cámara profunda, van a tener que ser adaptados al formato del láser, para conseguir homogeneizar los datos, ya que la cámara proporciona unos datos con una estructura y formato distintos.

Para entender las características de los datos proporcionados por el láser LDS-01, es importante recordar cómo se obtiene la información procedente de los sensores con el software ROS, a través de los mensajes en los *topics*. En relación a esto, ROS proporciona un paquete llamado *sensor_msgs.msg*, que dispone de distintos tipos de mensajes dependiendo del tipo de sensor que se esté utilizando, debido a que no será el mismo mensaje si son datos de distancia, de temperatura o de presión. Los datos, por su parte, pueden ser de distinto tipos numéricos; valores booleanos (*bool*), enteros (*int*), reales con coma flotante (*float*) o cadenas de bytes (*str*).

Para el láser LDS-01, cuando se recibe un *topic* procedente de él, el mensaje proporcionado por ROS es de tipo 2D conocido como *LaserScan*. A continuación, se muestra un ejemplo de un mensaje de tipo *LaserScan*, perteneciente al *topic agent1/scan*, que es el que lleva el mensaje con los datos de distancia del láser

```
---
header:
seq: 14727
stamp:
secs: 3098
nsecs: 430000000
frame_id: "agent1_tf/base_scan"
angle_min: 0.0
angle_max: 6.28318977356
angle_increment: 0.273182183504
time_increment: 0.0
scan_time: 0.0
range_min: 0.119999997318
range_max: 3.5
ranges: [3.832068920135498, 3.8560056686401367, 1.8473576307296753,
         2.0415027141571045, 2.286090135574341, 1.979326605796814,
         1.8840479850769043, 1.910698652267456, 2.130638599395752,
         2.3931479454040527, 1.9184242486953735, 1.710880994796753,
         1.6612005233764648, 1.7393394708633423, 1.9513779878616333,
         2.406503677368164, 0.6633331179618835, 0.5924382209777832,
         0.5996957421302795, 0.6123553514480591, 0.7239275574684143,
         0.9311634302139282, 4.117534637451172, 3.83327579498291]
intensities: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
              0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
              0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 0.0]
---
```

Como se puede apreciar la estructura del mensaje *LaserScan*, este está formado por distintos campos, de todos ellos, excepto el *header* que es un campo que suele estar en la mayoría de los mensajes que proporciona ROS que ayudada a identificar cada medida que se hace con el sensor, el resto de los campos son de tipo numérico *float32*, que es un float de 32 bits, por lo que en cada uno de esos nueve campos se proporciona un valor, que es un número real. Los dos últimos campos, *ranges* y *intensities*, no ofrecen un único valor, sino un rango de valores, debido que ahí es donde están todos los valores

leídos por el láser. A continuación, se explica qué información se aporta en cada uno de los campos que integran el mensaje *LaserScan:*

- *Header*: es la cabecera del mensaje, que sirve como identificación del mensaje que se ha recibido. Está formado por 3 subcampos:
  - *Seq*: número de secuencia del mensaje. Empieza en 0 y va sumando 1 en cada mensaje.
  - *Stam*: indica el tiempo, en segundos y nanosegundos, del mensaje respecto al primero que se ha emitido. La secuencia es muy importante para poder sincronizar diferentes mensajes.
  - *Frame_id*: especifica el punto de referencia del mensaje, de donde vienen los datos. En el código anterior se aprecia como viene del agente 1.
- Float32 *angle_min*: Indica el ángulo en radianes en el que inicia el scan
- Float32 *angle_max*: Indica el ángulo en radianes en el que finaliza el scan
- Float32 *angle_increment*: Indica la distancia angular entre los distintos rayos del láser. La división del valor del ángulo del láser entre este valor arroja el número de rayos que tiene el láser, es decir, el número de valores de distancia que va a dar el sensor.
- float32 *time_increment*: Tiempo entre medidas en segundos, utilizado para interpolar posiciones cuando el escáner es móvil.
- float32 *scan_time*: Tiempo entre cada scan, medido en segundos.
- float32 *range_min*: Mínimo valor en metros que puede leer el láser, al ser fundamental la evasión de obstáculos, se plantea este valor como 0.12 metros, debido a que con distancias más cercanas a 0 las medidas no son tan precisas.
- float32 *range_max*: Máximo valor en metros que puede leer el láser, establecido en 3.5 metros
- float32[] *ranges*: Vector que recolecta las distancias calculadas por cada uno de los rayos del láser, en este caso hay 24 datos.
- float32[] *intensities*: Vector que contiene la intensidad, o brillo, de los 24 rayos láseres reflejados.

Esta es la estructura que siguen los distintos campos que forman el mensaje *LaserScan*. Es importante destacar que excepto el *header, ranges* e *intensities,* el resto de los campos tienen valores fijos que sirven como información de las características que tiene el láser, y parte estos parámetros que aparecen en el mensaje se indican en un archivo tipo *xacro*, que es un lenguaje macro XML muy utilizado para la configuración de robots, mostrándose, a continuación, la parte de código *xacro* utilizado para establecer las características del láser. Como ya se ha comentado, se pretende que la medida del láser abarcase todo lo que rodea al robot, por eso en *angle_min* y *angle_max* el rango va de 0 a 6.28318 radianes, que representan los 360 grados. Por otra parte, se ha establecido que haya 24 valores, como se aprecia en código en la parte de *samples* y, por último, que el rango vaya de 0.12 a 3.5 metros, debido a que es la distancia máxima que puede medir el láser LDS-01 en la realidad.

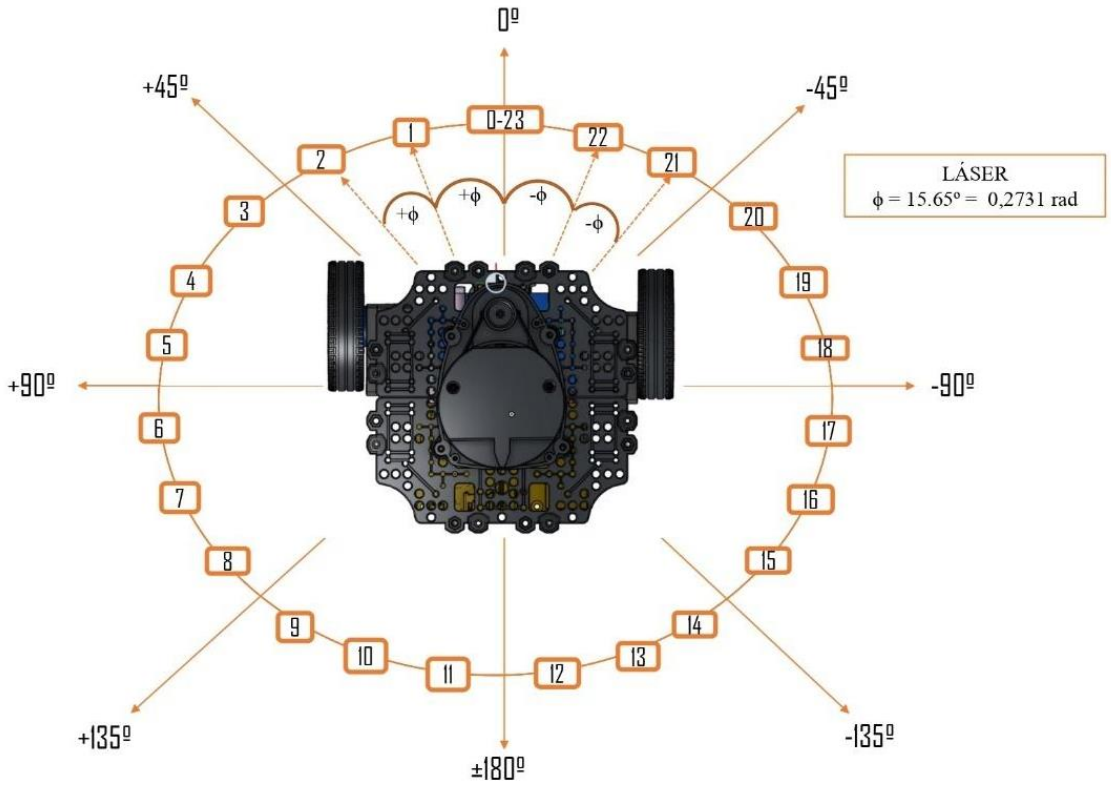
```
1  ...
 2 <ray>
 3   <scan>
 4     <horizontal>
 5       <samples>24</samples>
 6       <resolution>1</resolution>
 7       <min_angle>0.0</min_angle>
 8       <max_angle>6.28319</max_angle>
 9     </horizontal>
10   </scan>
11   <range>
12     <min>0.120</min>
```

```
13      <max>3.5</max>
14      <resolution>0.015</resolution>
15    </range>
16 </ray
17 ...
```

Tanto las *intensities*, que no resultan útiles para este trabajo, como *ranges*, van arrojando 24 valores en cada lectura. Respecto a este último, si la lectura de alguno de los rayos del láser es mayor a 3.5 metros, en el vector aparece el valor con un *string* que pone *inf*, lo que significa que hay una distancia mayor a la que el láser puede leer, para ello, se ha establecido que cuando aparece un *inf*, ese valor se transforma a 3.5 metros, para que todos los valores del vector sean número y poder trabajar correctamente. Por último, los 24 valores del que obtiene el láser LDS-01, son los 24 datos de entrada necesarios para las redes neuronales.

Por último, en la Figura 19 se aprecia como están distribuidos estos valores que obtiene el láser del robot. Como se aprecia, tanto el primer (posición 0) como el último valor (posición 23) apuntan hacia el mismo sitio, la parte más frontal del robot, esto es debido a la configuración propia del láser, que coloca esos dos rayos apuntando hacia prácticamente el mismo sitio, de esta manera se obtienen los valores de 24 rayos, aunque realmente sería como si fuesen 23 rayos ya que dos son prácticamente el mismo. En cuanto a la separación entre estos 23 rayos distintos, es de 0.2731 radianes, o 15.65º.

Figura 19 Distribución de los 24 valores de distancia proporcionados por el láser LDS-01.



4.3.2. Cámara de profundidad

La cámara escogida para utilizar en este trabajo es la cámara de profundidad Intel RealSense D435, que es una solución en formato pequeño (90 mm x 25 mm x 25 mm) y de bajo costo que presenta un amplio campo de visión, ofreciendo altas prestaciones en entornos de poca iluminación, así como en aplicaciones de movimiento rápido, tanto en ambientes *indoor* como *outdoor*, siendo una cámara muy utilizada en el campo de la robótica y de la realidad virtual [44].

Figura 20 Componentes de la cámara RealSense D435

La cámara de profundidad Intel RealSense D435 es de tipo estéreo y está compuesta, como se puede ver en la Figura 20, por dos sensores de profundidad, un módulo *Red Blue Green* (RGB) y un proyector IR, que juntos consiguen crear una estimación de la profundidad a través de detectar la luz infrarroja que ha sido reflejada en el objeto que tiene alrededor [45]. En cuanto a los sensores de profundidad, estos están desarrollados a partir de una tecnología estereoscópica de profundidad presentando una resolución de salida de hasta 1280 x 720 y una velocidad de fotogramas de profundidad de hasta 90 fps. En cuanto al módulo RGB, presenta una resolución de fotogramas de 1920 x 1080 con una velocidad de 30 fps [44].

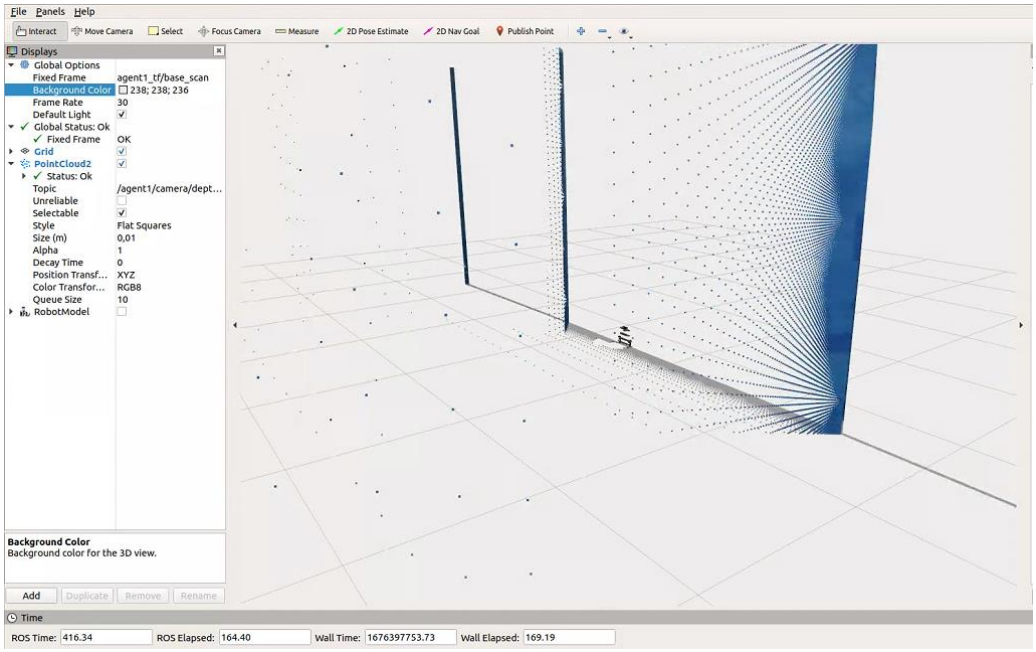
En cuanto a los datos ofrecidos por la cámara, estos son muy distintos a los del láser en formato y contenido, ya que, al igual que este mostraba un único *topic* llamado */scan* con el tipo de mensaje *LaserScan*, en el caso de la cámara son muchos, y variables, los *topics* que se muestran, dependiendo de las características y parámetros que tienen. En concreto, cuando se lanza por terminal el comando *rostopic list*, que es el encargado de mostrar todos los *topics* que hay activos, la cámara tiene asociado 16 *topics* distintos. El contenido de estos es muy variado también, destacando dos grupos principales, los *topics* que están relacionados con el módulo RGB de la cámara Intel RealSense D435, que aparecen como *agent1/camera/rgb/* y los que están relacionados con los sensores de profundidad, que son 3 y se muestran como *agent1/camera/depth.*

Uno de los *topics* más interesantes que se generan es el de *agent1/camera/rgb/image_raw* ya que es el que permite ver en *Rviz*, que es una herramienta de visualización 3D de ROS [46], lo que la cámara ve en color, es decir, en ese *topic* están los datos que permiten ver la transmisión de la cámara en directo, teniendo un tipo de mensaje *sensor_msg/Image*, que se caracteriza por ser mensajes que contienen imagen sin comprimir. También hay *topics* que dan información sobre las características de la cámara y de los parámetros de cada uno de los módulos de esta, como son los *topics* que acaban con */camera_info* o */parameter_descriptions.*

El *topic* más importante para este trabajo es el de *agent1/camera/depth/points* debido a que es el encargado de generar la nube de puntos con la que la cámara Intel RealSense D435 estima la profundidad y la distancia a los objetos que tiene a su alrededor. Este *topic* tiene el tipo de mensaje ROS de *sensor_msg/PointCloud2*¸ que se caracteriza por contener una colección de puntos N-dimensionales que contienen información sobre la profundidad de lo que rodea a la cámara. En la Figura 21 se puede apreciar un ejemplo de cómo es una nube de puntos para una cámara de 180 grados, en la que los puntos marcan los obstáculos que podría tener el robot a su alrededor. Por otra parte, a diferencia del láser y el mensaje *LaserScan*, que como se ha visto está perfectamente estructurado mostrando los 24 valores de

distancia, en el caso del mensaje *PointCloud2* no sucede así, ya que los datos se muestran sin procesar y no se muestran tan claros como en el caso del láser.

Figura 21 Captura de la herramienta Rviz con la nube de puntos del PointCloud2 procedente de la cámara.



Este último *topic* visto de *agent1/camera/depth/Points* es importante porque al ser el que determina la distancia con la cámara, será el utilizado para ser transformado al formato del láser, para que ambos tengan la misma estructura, en el siguiente apartado se explicará esto. Antes de continuar, es importante indicar que en este trabajo se va a trabajar con distintas configuraciones de la cámara para comprobar cuál es la más idónea y se acerca más al funcionamiento que tiene el robot cuando funciona a partir de los datos del láser, por lo que más adelante se verá cómo se van a utilizar diferentes cámaras con diferentes ángulos de visión para ver cómo afecta esto al comportamiento del robot. En el siguiente apartado, por su parte, se va a explicar de qué forma se consiguen transformar los datos de la cámara al formato válido del láser.

4.4. Homogeneización de los datos de la cámara y láser

Como ya se ha comentado, los datos proporcionados por la cámara utilizada no son adecuados para el entrenamiento de las redes neuronales, debido a que no presentan la misma estructura y formato que tiene el láser LDS-01, que es el sensor que se ha tomado como referencia a la hora de establecer como van a ser los datos de entrada de las dos redes. Debido a esto, es necesario poder transformar los datos proporcionados por las cámaras, siendo esto la parte principal del trabajo; convertir los datos heterogéneos en un formato común válido.

Para conseguir esto, se han convertido los datos de la cámara RealSense D435 al formato que presentan los del láser, que son los mensajes tipo *LaserScan*, y esto se ha realizado mediante un paquete disponible de la librería de ROS llamado *pointcloud_to_laserscan*. Este paquete convierte mensajes del tipo *sensor_msgs/msg/PointCloud2* en *sensor_msgs/msg/LaserScan*, transformando la nube de puntos 3D que se obtienen con la cámara RealSense D435 en un escaneo láser con datos 2D [47]. Con este paquete se consigue el objetivo planteado, en tanto que se consigue que dos sensores completamente distintos, cámara y láser, tengan el mismo formato en los datos, ambos con el tipo de mensaje *LaserScan*, de forma que se pueden usar los dos sensores sin problema para el entrenamiento de las dos redes neuronales.

El proceso de conversión se realiza a través de la creación de un nodo ROS con el paquete *pointcloud_to_laserscan,* en el que se indican los datos que se van a convertir y que características tendrán los datos de salida y el nodo se lanza a la vez que el robot, de manera que cuando el agente aparezca ya en el simulador, el proceso de transformación de los datos ya esté funcionando. El siguiente código es un ejemplo del nodo *pointcloud_to_laserscan* con el que se realiza la conversión de los datos.

```
<node pkg="pointcloud_to_laserscan" type="pointcloud_to_laserscan_node"
      name="pointcloud_to_laserscan_$(arg agent_name)">
    <remap from="/$(arg agent_name)/cloud_in"
           to="/$(arg gent_name)/camera/depth/points"/>
    <remap from="/$(arg agent_name)/scan"
           to="/$(arg agent_name)/camera"/>
    <rosparam>
        target_frame: agent1_tf/camera_link
        transform_tolerance: 0.01
        min_height: 0.0
        max_height: 1.0
        angle_min: -0.68539816339
        angle_max:  0.88539816339
        angle_increment: 0.2244
        scan_time: 1.0/30.0
        range_min: 0.00
        range_max: 3.5
        concurrency_level: 1
        use_inf: true
    </rosparam>
</node>
```

En el código anterior se puede apreciar la estructura que presenta el nodo *pointcloud_to_laserscan*, en la que primero están los *topics* implicados en el proceso de transformación y después los parámetros necesarios. Por parte de los *topics*, se tienen los siguientes:

- *Topics* suscritos: es la nube de puntos de entrada que va a que ser transformada, que provine de *sensor_msgs/msg/PointCloud2*
- *Topics* publicados: *topic* que se crea con el mensaje de salida con el formato *sensor_msgs/msg/LaserScan.*

El proceso de conversión se realiza a través de la etiqueta de ROS <*remap*> que sirve para reasignar etiquetas. Observando el código anterior, con el *remap* se transforma el *topic "agent1/camera/depth/Points",* que como se ha visto es el que tiene la nube de puntos 3D de la cámara, y lo transforma al formato *LaserScan* generando un nuevo *topic* definitivo con ya el mensaje transformado llamado "*/agent1/camera".* En cuanto a los parámetros necesarios para realizar la transformación, son los siguientes:

- *target_frame* (str): indica cual es el agente implicado en la transformación, o en caso de haber más de una cámara en el agente, cuál de ellas es la que se transformará. Debe hacer referencia a los *topics* suscritos.
- *transform_tolerance* (double): margen de tiempo que se da para localizar el *target_frame* para empezar la transformación.
- *min_height* (double): la mínima altura en metros de la nube de puntos que puede ser muestreada.

- *max_height* (double): la máxima altura en metros de la nube de puntos que puede ser muestreada.
- *angle_min* (double): indica el ángulo en radianes en el que inicia el scan, de la misma forma que sucedía con el *angle_min* del láser.
- *angle_max* (double): indica el ángulo en radianes en el que finaliza el scan.
- *angle_increment* (double): indica la distancia angular entre las distintas mediciones de cada escaneo, de la misma forma que con el láser, la división del ángulo de visión entre el *angle_increment* proporciona el número de datos de distancia que se obtienen.
- *scan_time* (double): la tasa de escaneo por segundos, estableciéndose siempre en 1/30.
- *range_min* (double): el mínimo valor en metros que puede tener una medida, estableciéndose en 0 metros.
- *range_max* (double): el máximo valor en metros que puede tener una medida, siendo de 3.5 metros como en el caso del láser.
- *concurrency_level* (int): indica el número de subprocesos que se utilizan para el procesamiento de las nubes de puntos a transformar. Se utilizará el valor por defecto 1.
- *use_inf* (boolean): cuando está en *True*, si se obtiene un valor más alejado que del rango máximo, entonces se le pone el valor del *range_max* + 1, si en cambio está como *False*, el valor que está más alejado del máximo aparece como *inf*.

Con todo esto se consigue la transformación de los datos, a través de este paquete proporcionado por ROS que hace adaptar los datos de la cámara como si fueran los datos proporcionados por un láser, consiguiendo el mismo formato para ambos sensores. Ahora bien, es necesario realizar más procesos para adecuar debidamente los datos de la cámara, debido a que con el paquete *pointcloud_to_laserscan* la cámara va a proporcionar mensajes tipo LaserScan, pero estos no van a proporcionar 24 valores como en el caso del láser, sino menos, debido a que el ángulo de escaneo del láser era de 360 grados, pero para el caso de la cámara no sucede lo mismo ya que el ángulo que abarca es considerablemente menor, por lo que se van a tener menos datos de distancia.

Como ya se ha comentado, como las redes neuronales profundas utilizadas ya están estructuradas de manera fija, es necesario que siempre los datos de entrada sean iguales, con 28 valores, siendo los 24 primeros los referidos a los datos de distancia equiespaciados que rodean al agente, por lo que, como la cámara otorga menos valores de distancia, va a ser necesario completarlos hasta llegar al número necesario. Ese es el siguiente paso por realizar en este trabajo, analizar cómo se han rellenado aquellos valores que no han sido proporcionados directamente con la cámara, al no disponer esta de un rango de visión tan amplio como el del láser LDS-01.

Antes de explicar este proceso, es necesario conocer los 5 tipos de configuraciones que se han establecido para la cámara. El motivo por el cual se ha decido a utilizar la cámara con diferentes configuraciones, es decir, con diferentes ángulos de visión, es para poder tener un análisis más completo sobre cómo influye la cámara utilizada, y comprobar que configuración se acerca más al funcionamiento ideal que tendría con el sensor más preciso y de referencia, el láser.

4.4.1. Cámara 90º.

La primera configuración por la que se ha optado es la que se asemeja más a una cámara convencional, por el ángulo de visión que ofrece, 90º. De todos los tipos, este es el que menos ángulo abarca y por lo tanto es el que va a proporcionar menos valores de distancia. En el código desarrollado, la parte donde se indica el ángulo de visión de la cámara es en el nodo de *pointcloud_to_laserscan*, en la parte de *angle_min, angle_max y angle_increment*, como se ha visto anteriormente. Tanto en esta configuración de la cámara como en el resto, la estructura y contenido del nodo *pointcloud_to_laserscan* es el mismo, como en el último código del nodo *pointcloud_to_laserscan*, manteniéndose igual tanto los *topics* involucrados como los parámetros, a excepción de los tres parámetros ahora comentados que

determinan el ángulo de visión. En la Tabla 1 se puede apreciar cuál es el valor de estos tres parámetros para cada una de las cuatro configuraciones de una cámara con diferentes ángulos de visión.

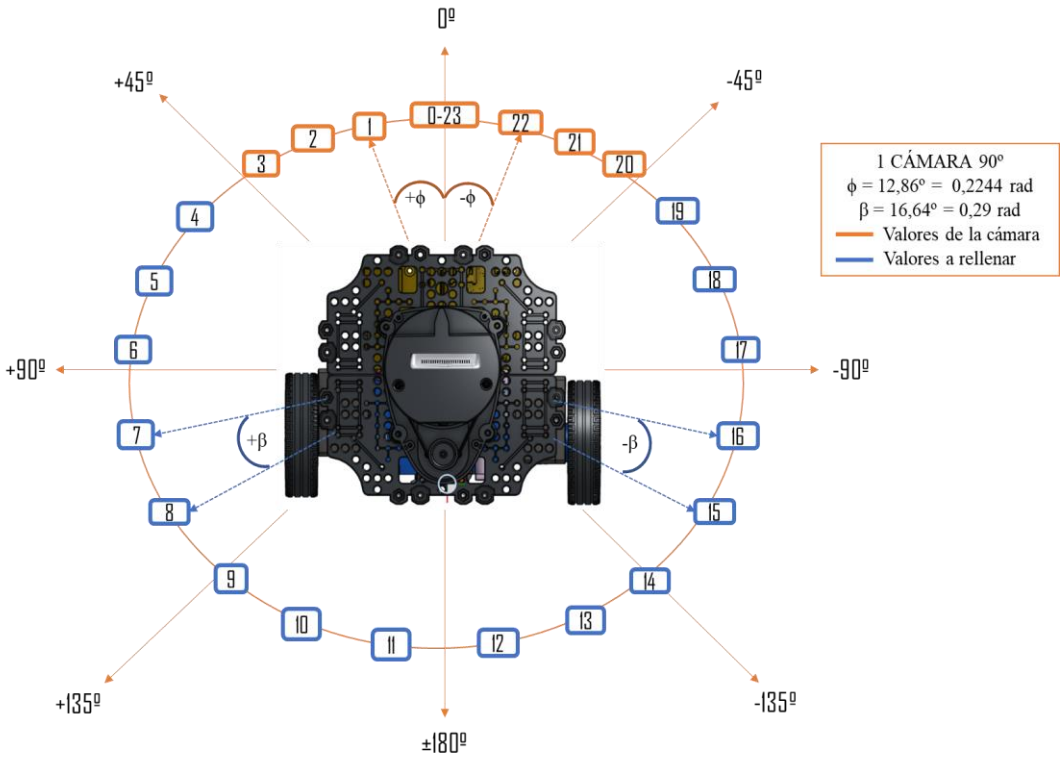
Tabla 1 Parámetros de angle_min, angle_max y angle_increment de las cuatro configuraciones de una cámara.

	Cámara 90º	**Cámara 120º**	**Cámara 150º**	**Cámara 180º**
angle_min (rad)	-0.68539816339	-0.9471975512	-1.208996939	-1.45
angle_max (rad)	0.88539816339	1.1471975512	1.408996939	1.65
angle_increment (rad)	0.2244	0.233	0.2381	0.2385

Para este primer caso de la cámara de 90º, el ángulo de visión en radianes, que es en la unidad que se trabaja siempre, es de 1.570796327 radianes. En la tabla, se puede apreciar como en todas las configuraciones el *angle_min* y *angle_max* no simétricos, es decir, para el caso de esta cámara, para ser simétrico el ángulo debería ir desde -0.785398 a 0.785398 radianes. Esto sería lo lógico, pero no está establecido así debido a que, por la forma en la que transforma el *pointcloud_to_laserscan* los datos, para que los haces de la cámara tengan una distribución más parecida a la del láser, el ángulo mínimo y máximo no deben ser exactamente simétricos. Esto sucede igual con todas las configuraciones.

En cuanto al *angle_increment*, como se ha comentado antes, si se divide el ángulo en radianes entre este valor, se obtiene el número de valores que se van a obtener de la cámara, en este caso son 7. Pero como en el láser el láser el primer y último valor del vector de 24 valores apuntaban al mismo sitio y tenían prácticamente el mismo valor, con las cámaras se va a hacer que suceda lo mismo, entonces el primer valor (posición 0) y último (posición 23) van a ser iguales, como sucede en el láser. De esta forma se tiene que con la cámara de 90º se obtienen 8 valores reales, mientras que los 16 restantes tendrán que ser rellenados, como se verá posteriormente en el apartado de relleno de datos. En la Figura 22 se puede apreciar como quedan distribuidos los 24 valores, y cuales se obtienen directamente de la cámara, los que están de color naranja, y cuáles será necesario rellenar, los de color azul.

Figura 22 Distribución de los 24 valores de distancia proporcionados por la cámara de 90º.

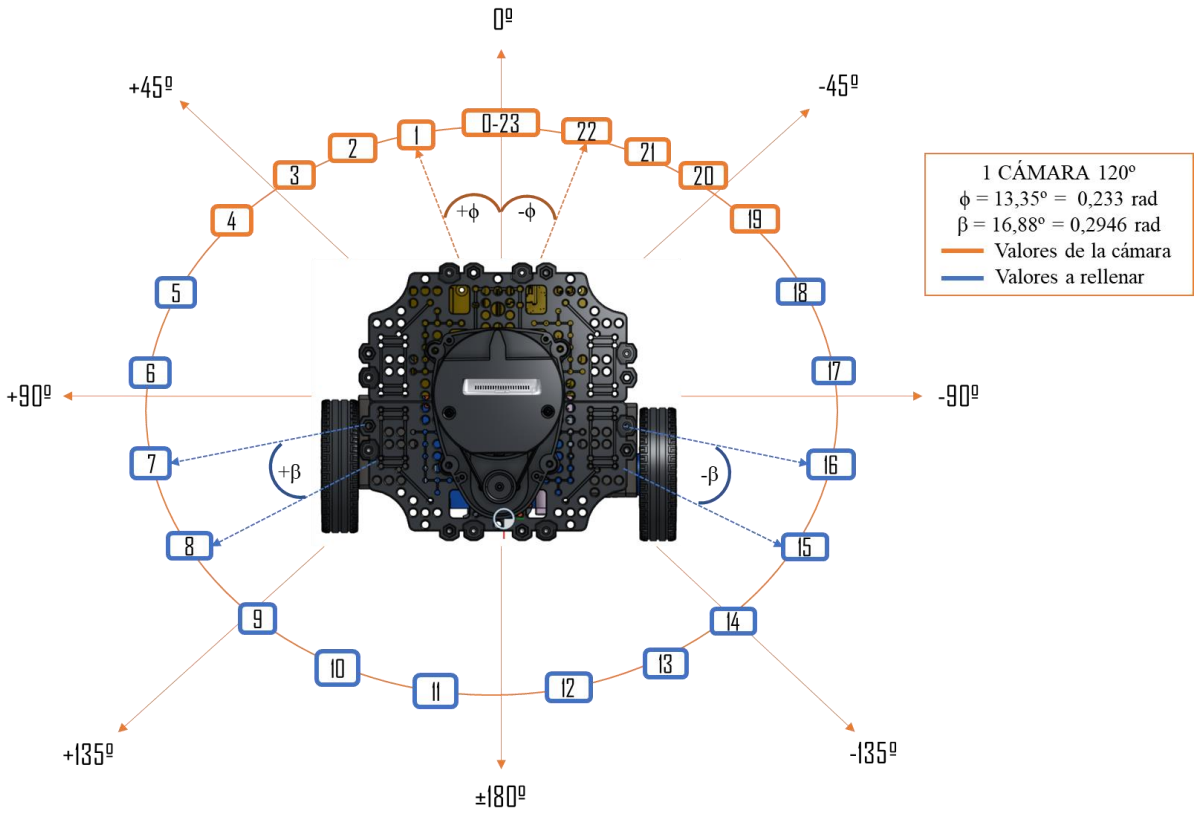


En cuanto al ángulo de separación entre los haces de la cámara, no va a ser igual a los que había con el láser, cada configuración de cámara tendrá una separación distinta como se puede apreciar en el *angle_increment* de la Tabla 1, ya que ese parámetro indica la separación entre los haces. No se ha querido forzar que la separación entre los valores sea exactamente igual a la del láser para comprobar que el entrenamiento puede funcionar correctamente, aunque los 24 valores de distancia no estén exactamente equiespaciados como en el láser, debido a que, al trabajar en la realidad con distintos sensores, aunque se intente acomodar los datos a un formato común, no siempre va a ser posible que sean exactamente iguales.

4.4.2. Cámara 120º

La segunda configuración de una cámara es con un ángulo de visión de 120º, siendo el rango del ángulo en radianes el que se aprecia en la Tabla 1. Esta configuración al tener una amplitud mayor, se van a obtener directamente desde el nodo *pointcloud_to_laserscan* de la cámara de 120º, un total de 9 valores reales, que al igual que se ha comentado para la configuración anterior, al establecer que el primer valor del vector va a ser igual que el último, con este tipo de cámara se obtendrán un total de 10 valores reales, siendo en este caso 14 los valores que será necesario tratar para darles un valor y poder completar el vector de 24. En la Figura 23 se puede apreciar la distribución de los diferentes valores.

Figura 23 Distribución de los 24 valores de distancia proporcionados por la cámara de 120º.

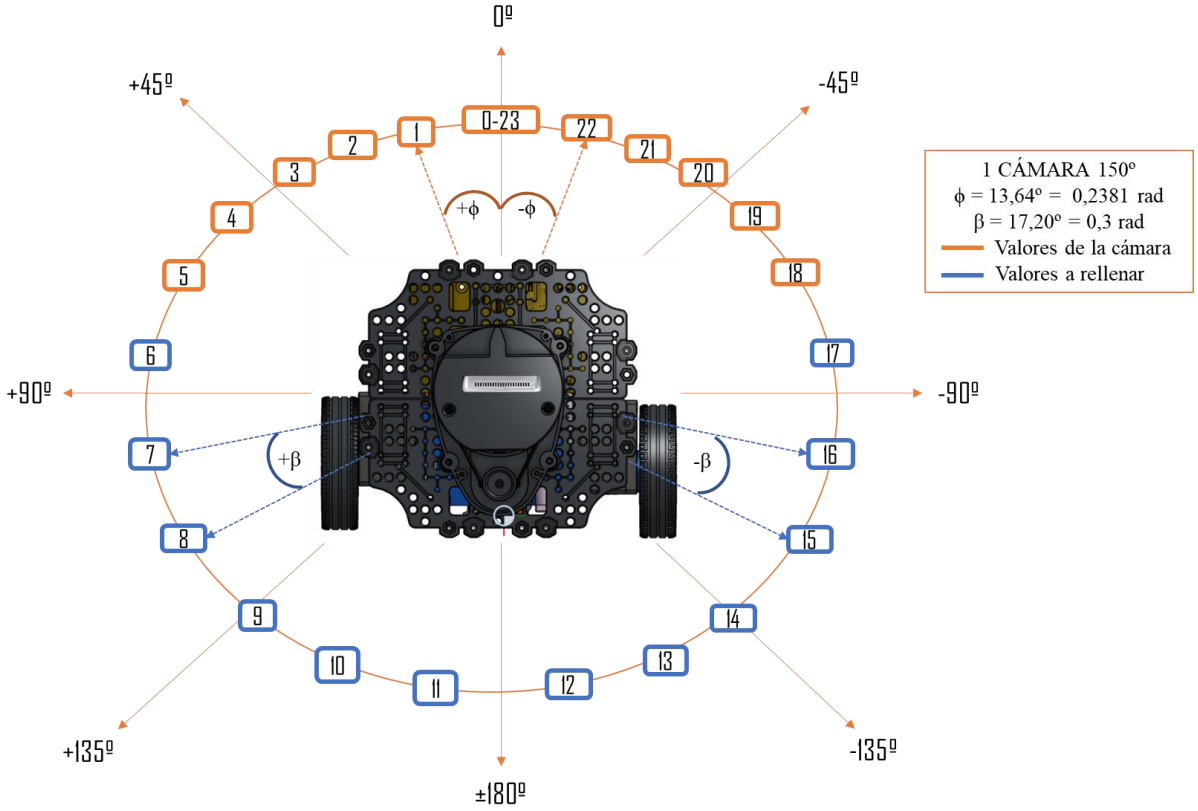


En cuanto al *angle_increment*, en esta configuración es de 0.233 radianes, que son 13.35º, por lo que al igual que con la cámara de 90º y el resto de las configuraciones, este valor es menor al del láser. Esta separación hace referencia a la separación entre los valores reales que son obtiene con la cámara, mientras que los valores a rellenar estarán separados a una distancia de 16.88º, tal y como se aprecia en la Figura 23.

4.4.3. Cámara 150º

En esta tercera configuración se tiene que el ángulo de visión es de 150º, 2.618 radianes, que están distribuidos como se observa en la Tabla 1. Con este tipo de cámara se obtiene dos valores reales más formando un total de 11, que finalmente se quedan en 12 reales y 12 que se será necesario rellenar. La separación entre estos valores es de 13.64º, mientras que la que hay entre los valores a rellenar es de 17.19º. En la Figura 24 se puede apreciar la distribución de los 24 valores de esta configuración de 150º.

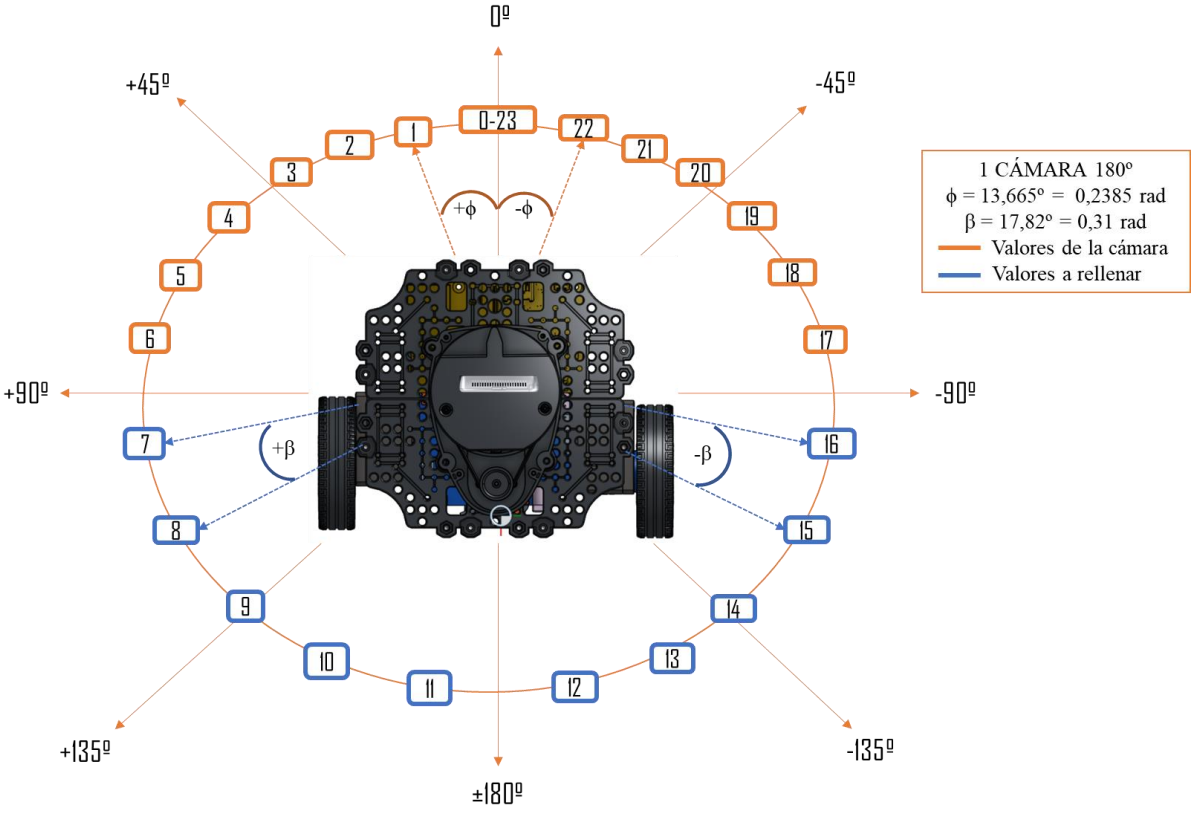
Figura 24 Distribución de los 24 valores de distancia proporcionados por la cámara de 150º.



4.4.4. Cámara 180º

La última configuración de una sola cámara es la de 180º, aunque realmente no llega a tener este valor, sino que es de 177.617º debido a que es el máximo valor con el que funciona bien la librería *pointcloud_to_laserscan*, debido a que cuando se introduce una amplitud de π radianes, que es lo que se pretendía, no se obtienen todos los valores necesarios. En este caso se van a obtener un total de 14 valores, por lo que en este tipo habrá menos valores a rellenar, 10, de los que se tienen reales, como se aprecia en la Figura 25.

Figura 25 Distribución de los 24 valores de distancia proporcionados por la cámara de 180º.



Estas serían las cuatro configuraciones realizadas con una sola cámara, como se ha visto, se ha intentado que la distribución de los haces tenga una forma similar a la del láser, siendo el primer y último valor iguales, el valor central, al igual que sucedía prácticamente en el láser, pero, por otra parte, no se ha buscado mantener la misma separación homogénea entre haces como sucedía en el sensor LiDAR, para ver si el entrenamiento funciona igual de bien. A continuación, se explica la última configuración que implica el uso de cámaras, aunque en este caso serán dos las que estén integradas en el robot.

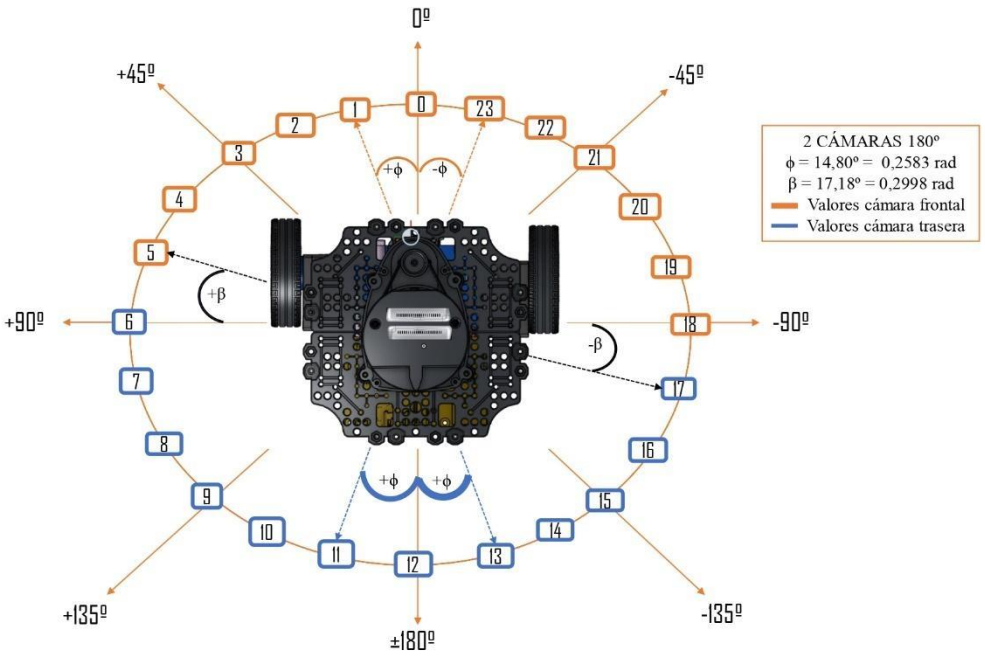
4.4.5. 2 cámaras 180º

Por último, se tiene la configuración de dos cámaras de 180º grados cada una, al igual que con la configuración anterior, realmente cada cámara es de 177.617º, que es lo máximo ángulo con el que se obtienen resultados óptimos. La configuración de los tres parámetros del *pointcloud_to_laserscan* para ambas cámaras es:

- *angle_min*: -1.55 radianes.
- *angle_max*: 1.55 radianes.
- *angle_increment*: 0.25833 radianes.

Con esta configuración lo que se busca es obtener los 24 valores directamente desde la cámara para no tener que rellenar datos, para ello, cada cámara va a proporcionar 12 valores; una los valores frontales y la otra los traseros. La principal diferencia de esta configuración con el láser es que, con este el primer y último valor eran prácticamente el mismo al apuntar hacia el mismo sitio en haz, pero con las 2 cámaras no, los 24 valores haces van a apuntar a una dirección diferente, para comprobar si afecta mucho el hecho de que los dos haces frontales no apunten al mismo sitio, como pasa en el láser, ya que esto no es lo más común.

Figura 26 Distribución de los 24 valores de distancia proporcionados por las dos cámaras de 180º.



En la Figura 26 se puede apreciar la distribución de los distintos haces, los de la cámara frontal en color naranja y los de la trasera en azul claro. Cabe destacar que los 12 valores de cada cámara están separados por 14.80º, mientras que la separación entre el último valor de una cámara y el primero de la otra es de 17.18º. Volviendo a la librería *pointcloud_to_laserscan*, para esta configuración ha sido necesario utilizar dos nodos de esta, una para cada cámara, con los mismos parámetros y cada una suscribiéndose a los *topics* o bien de la cámara frontal, o de la trasera. Por otra parte, al estar utilizando dos cámaras simultáneamente, es imprescindibles que estas estén sincronizadas para que el algoritmo funcione correctamente, es decir, ambas tienen que obtener cada lectura al mismo tiempo, de forma que los 24 valores de distancia que se obtienen de ambas, 12 valores procedentes de cada cámara, hayan sido tomados justo en el mismo instante de tiempo. En el siguiente apartado se explica cómo se ha conseguido esta sincronización de datos entre más de un sensor.

4.5. Sincronización de los datos

La sincronización de datos es un proceso que se implementa con el fin de integrar datos provenientes de diferentes sensores para estimar parámetros y estados necesarios, como es para el caso de este trabajo, la localización y orientación de un agente en un ambiente en función de los objetos que tiene a su alrededor. Este proceso garantiza que los datos sean precisos, seguros y actualizados sin incoherencias, errores y otras fallas que se puedan presentar, ya que, lamentablemente, incluso fallas menores pueden generar un impacto negativo significativo en la toma de decisiones [48].

Para conseguir esto, se va a utilizar el paquete de ROS llamado *message_filters*, el cual integra una serie de filtros que reciben mensajes generando otros nuevos en función de las condiciones especificadas. De todos los disponibles en el paquete *message_filters*, para realizar la sincronización de datos entre sensores, los filtros más adecuados son *ApproximateTime* y *ApproximateTimeSynchronizer*, que utiliza un algoritmo adaptativo para hacer coincidir dos *topics* suscritos en función de su marca de tiempo, pudiendo generar posteriormente un *topic* a partir de los dos sincronizados [49]. Ambos filtros tienen una estructura y funcionamiento realmente similares, siendo la diferencia principal entre ambos, que el segundo filtro se utiliza cuando alguno de los mensajes a filtrar no tiene contiene el campo de cabecera. Para la sincronización de las dos cámaras se ha optado por utilizar el filtro *ApproximateTime* debido a que los mensajes a sincronizar de ambas presentan cabecera y la misma estructura.

El proceso de sincronización de datos se realiza mediante un script de Python en el que se crea un nodo ROS llamado *camera_sync*, en el que, primero, se suscriben los *topics /camera* y */back_camera*, correspondientes a los 12 valores generados por la cámara frontal, y los 12 de la trasera, respectivamente, que han sido previamente generados por el paquete *pointcloud_to_laserscan* como ya se ha explicado. En el siguiente código se muestra cómo se realiza la suscripción de los *topics /agent1/camera* y */agent1/back_camera*en para empezar el proceso de sincronización:

```
1 ...
2 front_sub = message_filters.Subscriber("camera", LaserScan)
3 back_sub = message_filters.Subscriber("back_camera", LaserScan)
4 ...
```

Una vez se tienen los *topics* suscritos, se procede a la aplicación del filtro de sincronización *ApproximateTime* con el siguiente código:

```
1 ...
2 pub = rospy.Publisher('camera_sync', LaserScan, queue_size=1)
3 ats = message_filters.TimeSynchronizer([front_sub, back_sub], queue_size=1)
4 ats.registerCallback(callback)
5 ...
```

En la primera línea de código se genera el *topic* donde se van a publicar los 24 valores, ya sincronizados, de la distancia a los objetos que rodean al agente. Este *topic* se va a llamar *camera_sync*, y para publicar en él será necesario utilizar la siguiente orden *pub.publish(data)* siendo *data* los valores sincronizados. La segunda línea es el filtro propiamente dicho, en el que se llama a ambos *topics*, indicándole que el tamaño máximo de la cola de mensajes salientes debe ser de 1, sincronizando de esta forma los *topics /camera* y */back_camera*.

Una vez sincronizados en el tiempo, con la tercera línea se llama a la función *callback* que es la encargada de recibir en cada lectura de las cámaras, los 12 valores de cada una, ordenarlos para que estén correctamente como en la Figura 26 y publicar el *topic camera_sync* con los 24 valores perfectamente sincronizados y ordenados, ya que serán esos los datos que utilice el robot en las redes neuronales cuando esté funcionando con la configuración de 2 cámaras de 180º.

De esta forma se consigue la sincronización de los datos cuando proceden de dos sensores, en este caso de dos cámaras. Es importante destacar, que, en este trabajo, el proceso de sincronización también se utiliza cuando se el robot funciona con una sola cámara, y esto es debido a que, aunque el robot funcione únicamente con los datos obtenidos por la cámara, se hace una sincronización con los datos del láser, con el fin de comprobar que las colisiones que detecta el robot con la cámara sean las mismas que con el láser.

A diferencia de con las dos cámaras que se utiliza el filtro *ApproximateTime*, para este caso se utiliza el filtro *ApproximateTimeSynchronizer* debido a que, aunque tanto el láser como la cámara después de la homogeneización de datos presentan la misma estructura de datos, con este filtro se obtienen mejores resultados que con el filtro utilizado para las dos cámaras. Los parámetros de *ApproximateTimeSynchronizer* son los mismos que para *ApproximateTime* salvo que se añade un parámetro más que es el *slop*, que indica el retardo máximo en segundos que puede haber para sincronizar los mensajes. A continuación, se muestra el código donde se suscriben los *topics* de la cámara y el láser, se crea el *topic* donde se publicarán lo datos sincronizados, *camera_laser_sync*, se aplica el filtro *ApproximateTimeSynchronizer* y se llama a la función *callback* que hace todo el proceso de sincronización y ordenación de los datos:

```
1 ...
2 camera = message_filters.Subscriber("camera", LaserScan)
3 laser = message_filters.Subscriber("scan", LaserScan)
```

```
pub = rospy.Publisher('camera_laser_sync', LaserScan, queue_size=1)
ats = message_filters.ApproximateTimeSynchronizer([camera, laser],
      queue_size=2, slop=0.15)
ats.registerCallback(callback)
...
```

La sincronización entre cámara y láser se realiza para que los datos de aciertos (alcanzar el objetivo) y fallos (colisiones) del robot sean completamente reales, y que la información que se extraiga de los entrenamientos utilizada en el análisis de los resultados sea lo más real posible. Como se acaba de comentar, esto se realiza simplemente como comprobación, ya que tanto el proceso de entrenamiento como el cálculo de la acción a realizar por el robot, se realiza únicamente con los datos obtenidos por la cámara. Para este caso de sincronización de cámara y laser, los *topics* a los que se suscribe en este caso son */camera* y */scan*, respectivamente, y el *topic* que se crea se llama *camera_laser_sync*, que en este caso contiene 49 valores:

- Primeros 24 son los valores de distancia procedentes de la cámara, utilizados para el entrenamiento y cálculo de la acción.
- Siguientes 24 valores son los de distancia procedentes del láser, utilizados únicamente para la comprobación de las colisiones del agente.
- Último valor que indica el tipo de cámara que se está utilizando, si de 90, 120, 150 o 180 grados, ya que esta sincronización entre cámara y láser se realiza para las cuatro configuraciones con una sola cámara.

Ese último valor que identifica el tipo de cámara empleada se utiliza a nivel interno para el determinar cómo llevar a cabo el proceso de relleno de datos, que es lo que se explica en el siguiente apartado, que da por finalizado todo el proceso de homogeneización, sincronización y adecuación de los datos.

4.6. Relleno de datos

Como se ha visto en la descripción de las configuraciones de las cámaras, para el caso en el que solo hay una de ellas utilizándose, es necesario realizar un proceso de relleno de datos, debido a que como todas tienen un ángulo de visión menor al del láser, generan menos datos de lo necesarios. En ningún caso de las configuraciones vistas con una sola cámara, estas proporcionan más de 14 valores, por lo que el resto de los valores hasta llegar a 24 deben ser rellenados para que las redes neuronales puedan ser utilizadas.

Al no ser posible obtener los 24 datos reales de las medidas de distancia con las diferentes configuraciones de la cámara, se llevarán a cabo una serie de cálculos sobre los datos recolectados con el objetivo de obtener una aproximación que sirva de orientación de la distancia de los objetos que se encuentran en el entorno del robot.

El proceso de relleno se va a implementar en el código encargado de realizar las funciones del agente, en la parte de leer los datos procedentes de los sensores, que se va a realizar con una función llamada *scan_data()*, cuya labor consiste en obtener constantemente los datos del sensor utilizado, bien sea el láser, una cámara o dos cámaras. Es decir, el objetivo de la función *scan_data()* es obtener los 24 valores de la distancia del robot con su entorno, para guardarlos en una variable, que, junto con valores de distancia y orientación al objetivo y el valor de la distancia a la pared , conformarán los 28 valores que son el estado del robot, que se envía a la nube 1 para entrenar las redes neurales y a la nube 2 para que calcule la acción a realizar por el agente.

Esta función *scan_data()* está dividida en tres partes, cada una encargada de recibir los datos en función de si el sensor utilizado por el robot es el láser, las dos cámaras a la vez o una sola cámara con

alguna de las cuatro configuraciones dependiendo del ángulo de visión. La forma de recibir estos datos es a partir de la función *wait_for_message()* de la librería *rospy*, que se encarga de crear una suscripción a un *topic*, recibir un mensaje de ese *topic* y después cancelar la suscripción, para repetir este proceso en cada acción que realiza el robot. En la siguiente línea de código se muestra la parte de la función *scan_data()* donde se utiliza *wait_for_message()*, en este caso cuando se está utilizando el láser:

```
1 ...
2 data = rospy.wait_for_message('/'+self.agent_name+'/scan', LaserScan, timeout=5)
3 ...
```

Como se aprecia en el código anterior, la función *wait_for_message()* devuelve un variable, en este caso los datos procedentes del sensor, y tiene tres parámetros, que son los siguientes:

- Nombre del *topic*. En este primer parámetro se indica el *topic* del cual se va a extraer la información. El código ejemplo muestra como es este primer parámetro si el sensor del robot es el láser, /scan, si fuera dos cámaras, el *topic* sería el que se crea después de la sincronización de ambas, *camera_sync*, y, por último, si el sensor utilizado es alguno de los de una sola cámara, el *topic* del cual se extrae la información sería *camera_laser_sync.*
- Tipo de *topic*. En este segundo parámetro se indica cual de que tipo es el *topic*. Para los tres casos posibles de este trabajo, siempre va a ser *Laserscan*, porque después de la transformación mediante *Pointcloud_to_Laserscan* de las cámaras, tanto estas como el láser tienen el mismo tipo de mensajes.
- *Timeout*. Tiempo de espera en segundos. Si se excede ese tiempo esperando para recibir el mensaje del *topic*, se genera una excepción.

Una vez se reciben los datos de la distancia del agente y su entorno, se pasa al tratamiento de los mismo, si es necesario. Como ya se ha visto, en el caso del láser no es necesario modificar ya que son los datos que se han tomado como referencia para establecer la estructura, con sus 24 valores. Para el caso de las dos cámaras de 180º tampoco es necesario modificar nada porque se tienen los 12 valores de cada una de las cámaras, estando ya ordenados esos valores correctamente durante el proceso de sincronización de los datos. Por último, es en las cuatro configuraciones con una sola cámara en la que sí hace falta modificar los datos de entrada para poder completar los 24 valores necesarios. A modo de recuerdo, estos son los valores reales que se obtienen con cada una de las configuraciones de las cámaras (los valores obtenidos de la cámara se van a denominar valores o datos reales, y los que se rellenan, valores artificiales):

- 1 cámara 90º. Se obtienen 8 valores reales.
- 1 cámara 120º. Se obtienen 10 valores reales.
- 1 cámara 150º. Se obtienen 12 valores reales.
- 1 cámara 180º. Se obtienen 14 valores reales.

El proceso de relleno es progresivo, es decir, la primera lectura que haga el robot no va a tener todos los valores artificiales calculados, sino que en cada paso que el robot ejecute, irá calculando valores artificiales, debido a que estos se van a calcular a partir de los valores reales inmediatamente anteriores. Por esto, va a existir una memoria que almacena los 24 valores de distancia de la acción anterior, al ser necesarios para el cálculo de los valores artificiales de la lectura actual.

Siguiendo esta fórmula, los valores artificiales se van calculando con cada paso que da el robot, de forma que hay momentos que el robot cuando lee los datos de la cámara, de los 24 valores, tendrá los reales obtenidos directamente por el sensor, algunos valores artificiales calculados a partir de los datos anteriores, pero habrá algunos valores que, como robot no ha dado suficientes pasos, no podrán haber sido calculados artificialmente, por lo que esos valores, que no son ni reales ni artificiales, se les va a llamar valores promedio y van a tener un valor fijo calculado a partir de los valores reales. Este valor fijo no ha sido calculado a partir de los valores anteriores, y es completamente un valor de relleno, que tendrá un valor entre 1 y 2. Esto se realiza así debido a que cuando no es posible rellenar todos los

valores restantes con valores artificiales calculados porque no hay información suficiente, es necesario rellenarlos con algún valor hasta obtener los 24. Para sintetizar, estos son los tres tipos de valores que se van a tener con las lecturas del robot cuando utiliza una cámara:

- Valores reales. Aquellos que se obtienen directamente desde la cámara después de aplicar el paquete *pointcloud_to_laserscan*. Pueden ser 8, 10, 12 o 14 valores en función del ángulo de visión de la cámara, y se actualizan en cada paso que da el robot con los valores que lee la cámara.
- Valores artificiales. Aquellos que se calculan matemáticamente a partir de los valores reales de la lectura anterior del robot.
- Valores promedio. Aquellos que aún no han podido ser calculado como los artificiales y que son un valor que se utiliza simplemente como relleno para poder completar los 24 datos de distancia necesarios.

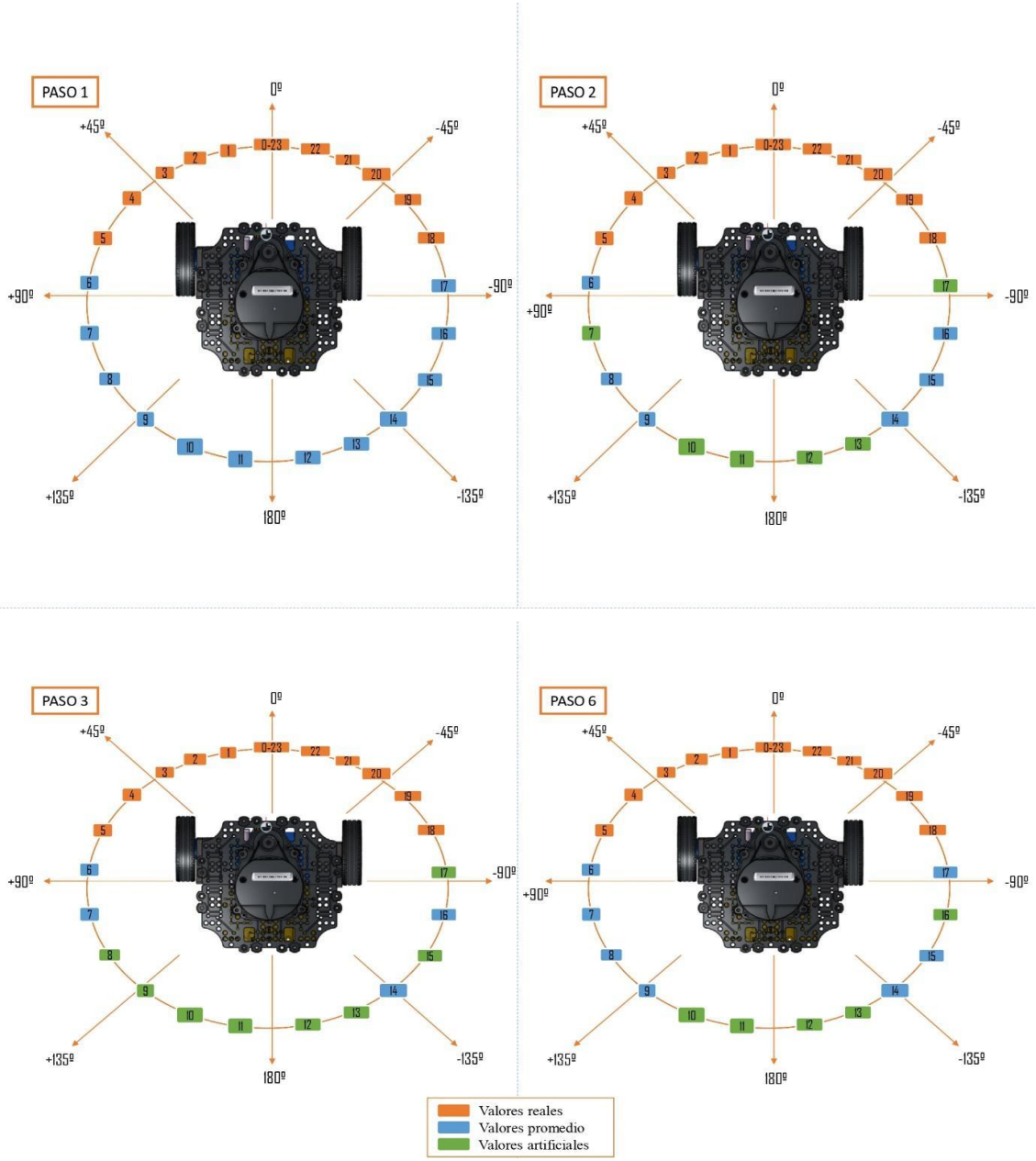
Para establecer cuál es este valor promedio en cada lectura, se realizaron diferentes pruebas en las que se establecía un valor determinado para ver cómo se comportaba el robot con cada uno de ellos. Se probó con que fuera un valor menor que 1, para que el robot interpretase que siempre tiene obstáculos cerca por detrás, también que el valor promedio sea siempre el valor máximo de 3.5 metros, para que pensase que tiene total libertad para girar. También se probó con valores mayores que 2, y por último, con valores comprendidos entre 1 y 2. De todas estas opciones, con la que mejor funcionaba el robot, es con la última, por lo que, para establecer este valor promedio en cada paso, de todos los valores reales recibidos por la cámara, se establecía como valor promedio el mayor de los valores que estuviese entre 1 y 2, y en caso de que no hubiese ningún valor real recibido entre ese rango, entonces el valor promedio, sería la media de los valores reales recibidos, forzando siempre que esté entre 1 y 2. El final objetivo es que, dentro del vector de 24 valores, haya el mínimo número de valores promedio, por eso se calculan los valores artificiales, que son más cercanos a las distancias reales.

Más adelante se explicará en detalle cómo se calculan estos valores artificiales, pero antes, para entender mejor el funcionamiento general del proceso de relleno de datos, se va a poner un ejemplo, suponiendo que se está utilizando la cámara de 150º grados y el robot se está moviendo hacia delante constantemente (acción 2). Es importante indicar que cada vez que se obtienen los valores de distancia con la función *wait_for_message()*, esos valores se guardan en la variable *scan_data* y se comienza el proceso de cálculo de los valores artificiales a partir de los valores de la lectura del paso anterior, que están guardados en la variable *scan_data_past.* De esta forma, cuando acabe el proceso de relleno de datos, en *scan_data* están los valores finales que se pasan a las dos nubes para el entrenamiento y cálculo de la acción. De esta forma el proceso que se sigue es el siguiente:

1. Cuando se realiza la primera lectura del robot, del vector de 24 valores de distancia, se tiene 12 valores reales calculados por la cámara, y al no tener ninguna lectura anterior con la que calcular los valores artificiales, los restantes 12 valores son promedio, como se aprecia en el paso 1 de la Figura 27.
2. Después de realizar el primer paso, se guardan los 24 valores de distancia del paso anterior y se obtiene la segunda lectura. De esta, se tienen los 12 valores reales calculados directamente por el sensor, y ahora el *scan_data[7]* y *scan_data[17],* van a ser valores artificiales que se han calculado a partir de alguno de los valores reales del *scan_data_past*, posiblemente del *scan_data_past[5]* y *scan_data_past[18],* respectivamente, que eran los últimos valores reales calculados por la cámara en el paso anterior.
   Asimismo, los valores las posiciones 10, 11, 12 y 13 del vector *scan_data* también son valores artificiales, que al ser los valores traseros tiene otro método de cálculo que se explicará más adelante. Por otra parte, el restante de valores del vector son valores promedio, como se aprecia en el paso 2 de la Figura 27.

3. En el tercer paso se tiene los 12 valores reales leídos por la cámara junto con los nuevos valores artificiales *scan_data[9]* y *scan_data[15]*, calculados a partir del *scan_data_past[7]* y *scan_data_past[17]*, que en el paso anterior no eran valores reales, pero sí eran valores artificiales calculados a partir de los reales. También se tiene el valor artificial *scan_data[17]* que se ha vuelto a calcular con el *scan_data_past[18]*, pero ahora ya no está *scan_data[7] sino el scan_data[8]*, calculado a partir de *scan_data_past[5]*.
   Las posiciones 10, 11, 12 y 13 del vector siguen siendo valores artificiales calculados a partir de esas mismas posiciones del vector *scan_data_past*, y el resto de los valores son de relleno, como se aprecia en el paso 3 de la Figura 27.
4. Esto se repite constantemente y en cada paso son diferentes los valores artificiales que se rellenan dependiendo de los cálculos matemáticos, que se explicarán a continuación, paso 4 de la Figura 27 se aprecia otro posible caso que es que en el siguiente paso se obtiene únicamente el valor artificial de *scan_data[16]* en el lado derecho, mientras que en el lado izquierdo no se ha podido calcular ningún valor artificial a partir de los valores reales.

Figura 27 Proceso de relleno de datos para un agente con una cámara de 150º ejecutando la acción 2.



Como se ha podido apreciar, el proceso de relleno no es uniforme, en cada paso son distintos los valores que se rellenan, y esto es debido a los cálculos matemáticos realizados y a la cercanía del robot a los obstáculos, ya que en algunas ocasiones, por ejemplo, con el quinto haz de la cámara se podrá obtener el valor artificial que se obtendría con el séptimo haz, otras veces el que se obtendría con el sexto haz, mientras que algunas otras veces no se podrá rellenar ningún valor, como se ha podido apreciar en la Figura 27.

En el ejemplo anterior se ha visto cuando el robot se está moviendo únicamente con velocidad lineal, es decir, no realiza ningún giro, en ese caso se rellenan con valores artificiales los valores traseros (posiciones 10, 11, 12 y 13 del *scan_data*) y pueden ser valores artificiales tanto las posiciones de la parte izquierda (las posiciones que se encuentran desde el 0 del vector *scan_data* hasta el 11) y los de la parte derecha (de la posición 12 del vector a la 23). Cuando hay giros hacia los lados no sucede igual, debido a que los valores traseros no se pueden calcular de la forma como se hace cuando se realiza la acción 2. También sucede que, cuando se gira hacia un lado, se van a rellenar con valores artificiales principalmente los valores de un lado solo, es decir cuando el robot gira a la derecha se van a poder obtener valores artificiales de la parte derecha principalmente, aunque si se encadenan varios giros consecutivos hacia la derecha, es posible que también se obtengan algún valor artificial de la parte más alta de la derecha, es decir, las posiciones 12, 13, 14 o incluso 15 del *scan_data*. Lo mismo, pero de forma inversa ocurre con los giros a la izquierda.,

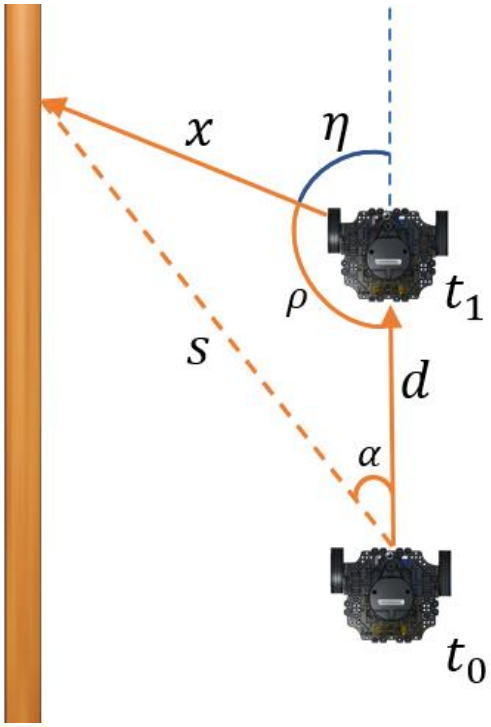
Para sintetizar, la clave del proceso de relleno de datos realizado en este trabajo consiste en, a partir de los datos de la lectura anterior, *scan_data_past*, utilizar cada uno de los valores reales, y los artificiales calculados a partir de los reales anteriores, para calcular los nuevos valores artificiales y determinar ese valor artificial a que haz se asigna, es decir, a que posición del *scan_data* corresponde ese valor artificial calculado. Con el método desarrollado se consigue en cada paso tener valores artificiales que son prácticamente iguales a los que serían si se obtuvieran de forma directa por la cámara, de esta forma, es posible que no se consiga que todos los valores del *scan_data* sean artificiales, pero sí que todos aquellos que lo son, sean lo más cercanos a la realidad posibles.

Por último, solo queda explicar cuáles son los cálculos matemáticos que se han realizado los procesos de relleno de valores, que se pueden separar en dos grupos para entenderlos mejor; los cálculos matemáticos para acciones de movimiento rectilíneo y cálculos para acciones de giro.

4.6.1. Cálculos matemáticos en acciones de movimiento lineal.

Como se ha visto, cuando se encadenan movimientos en recto hacia delante, acción 2, se realizan unos cálculos matemáticos al rellenar los datos, tanto para los valores laterales, tanto como para los valores traseros, como se ha explicado en el ejemplo de la cámara de 150º, siendo en ambos casos, cálculos realizados a partir de la trigonometría. Para el caso de los valores laterales, en la Figura 28 se puede apreciar un ejemplo gráfico de como se ha realizado la estimación de los valores, en este caso, para un valor del lado izquierdo.

Figura 28 Cálculos matemáticos para el relleno de datos en acciones rectilíneas.



En la Figura 28 se aprecia el movimiento de un agente en línea recta, que pasa de estar en t_0 a estar en t_1, cuando está en $t_1$es cuando se realizan los cálculos para determinar valores artificiales. El proceso que se sigue, junto con las fórmulas matemáticas utilizadas es el siguiente:

1. t_0 es la posición de partida del robot, y se va a suponer para entender mejor el ejemplo, que no ha realizado ninguna acción antes, por lo que es la posición inicial. El robot en esta primera posición lee los datos del entorno y los guarda en la variable *scan_data*.
2. El robot ejecuta la acción 2 y avanza hasta la posición t_1, y guarda los valores de la lectura anterior en la variable *scan_data_past*. Luego procede a leer los valores de la cámara, para guardarlos en la variable *scan_data.*
3. Una vez obtiene los valores de la distancia se procede con el relleno de datos. En el caso de la cámara de 150º, serán 12 valores reales que se obtienen directamente con la cámara, por lo que los restantes 12 valores se rellenan inicialmente con el valor promedio calculado en función de los valores reales, como se ha explicado antes. Una vez realizada esta primera estimación, en los pasos siguientes se procederá a actualizar los valores promedios para obtener los valores artificiales a partir de los valores reales leídos por el sensor en el paso anterior.
4. Para calcular los valores artificiales de la parte izquierda, se van a analizar los 6 valores reales de la parte izquierda. Usando trigonometría se calcula si alguno de los puntos leídos en el paso anterior puede corresponder a uno de los datos a estimar. Se va a suponer que se hace el cálculo con el sexto valor, que es el *scan_data_past[5]*.
5. En primer lugar, el valor del sexto haz obtenido en t_0, correspondiente al el *scan_data_past[5]*, es la s en la Figura 28 y el ángulo α es el del haz del *scan_data_past[5]*, que en este caso es 68.2º. Por su parte, el valor d es la distancia que recorre el robot, que se calcula con la velocidad lineal del robot y el tiempo que ha tardado en realizar la acción, es decir el tiempo que ha pasado entre t_0 y t_1:

$$d = vel_{lineal} * t_{acción} \quad (5)$$

6. Teniendo s, d y α se puede calcular con trigonometría tanto la x, que es el valor artificial calculado, como la ρ, que es un ángulo que se va a utilizar para determinar a qué haz corresponde este valor artificial calculado. En primer lugar, se calcula este valor x:

$$x = \sqrt{s^2 + d^2 - (2 * s * d * cos\ cos\ \alpha)} \quad (6)$$

7. Una vez se tienen todos los lados del triángulo, se calcula ρ de la siguiente forma:

$$\rho = \left(\frac{d^2 + x^2 - s^2}{2 * d * x}\right) \quad (7)$$

8. Por último, el ángulo suplementario de ρ es η, que sería el ángulo del haz que pertenece el valor artificial x calculado, es decir, si por ejemplo se ha obtenido que η es 120º entonces se busca que haz del *scan_data* tiene el ángulo más cercano a ese valor y se establece que ese haz tiene el valor artificial calculado. Siguiendo con el ejemplo, el *scan_data[8]* tiene un ángulo de 119.8º, siendo el más cercano a todos al valor calculado η de 120º, por lo que después de realizar todos los cálculos se ha obtenido que en t_1 el *scan_data[8]* tiene el valor artificial de x.

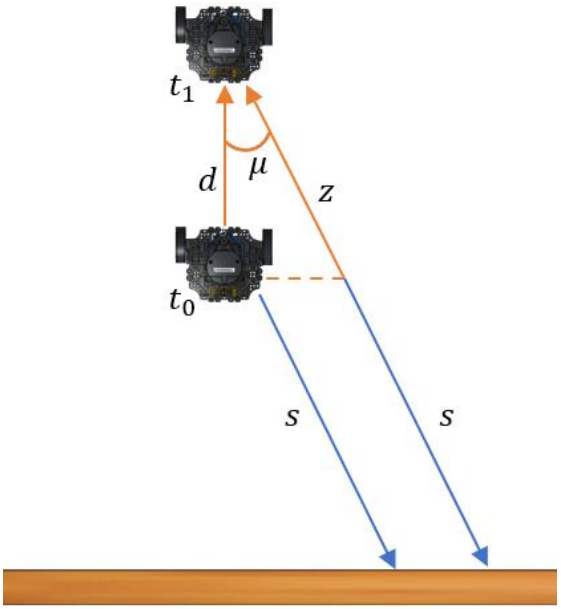
$$\eta = \pi - \rho \quad (8)$$

Este es el cálculo matemático realizado para determinar los valores artificiales cuando se realizan acciones con velocidad lineal únicamente, y como se ha visto con el ejemplo, con este método, se habría conseguido poder determinar el *scan_data[8]* a partir del *scan_data_past[5].* El proceso explicado, en este caso para el valor real del haz 5, se haría con el resto de los valores reales, por lo que los pasos del 4 al 8 se repetirían con cada uno de los valores reales. Por otra parte, esto ha sido un ejemplo para rellenar los valores de la parte izquierda, para los de la parte derecha sería igual salvo que los valores reales que se utilizan son los de la derecha, es decir, para 150º, de la posición 23 a la 18 del *scan_data.*

Es importante destacar que muchas veces no se pueden rellenar valores, y esto es debido a que cuando se calcula el valor artificial y se determina el haz al que corresponde, se tiene que ese haz es el mismo que con el que se ha calculado. Por ejemplo, se realizan los cálculos con el valor real de antes, *scan_data_past[5],* y después de realizar los cálculos matemáticos se obtiene que el haz que más cerca está del calculado ángulo η es el propio *scan_data[5],* por lo que no se rellena nada, porque el valor del *scan_data[5]* ya lo está dando directamente la cámara.

De esta forma es como se obtendrían los valores artificiales de los laterales cuando se están realizando movimientos lineales. Por otra parte, también se ha visto que para este caso también se podía calcular los 4 valores traseros, que son las posiciones 10, 11, 12 y 13 del *scan_data*. Es importante destacar que, al igual que para el caso de los valores laterales, estos valores se calculan inicialmente a partir de los valores reales, para el caso de los traseros esto no es así, sino que se calculan a partir de los valores promedio, debido a que resulta imposible hacerlo de otra forma. El objetivo de realizar esto, es que, aunque el valor inicial con el que se realizan los cambios no sea real, si el robot se está desplazando hacia delante, los objetos que pueda tener detrás se van a ir alejando del robot, por lo que, con los cálculos siguientes, los valores calculados también serán cada vez mayores, entonces, aunque los valores que se estén calculando no partan de valores reales, sí que serán cada vez mayores, haciendo ver al robot que se está alejando del obstáculo.

Figura 29 Cálculos matemáticos para el relleno de datos de los valores traseros en acciones rectilíneas.



Los pasos a seguir para este ejemplo de la Figura 29 son los siguientes:

1. Los valores a analizar aquí son los de la posición 10, 11, 12 y 13 del vector, pero par este ejemplo se va a suponer que se está realizando para el valor 13, por lo que en la Figura 29 el valor s sería en t_0 el *scan_data_past[13]* y el valor artificial que se busca calcular es en t_1 el *scan_data [13]*, que sería la suma de s más z.
2. El ángulo μ es el que forma ese haz con respecto al ángulo 180º del robot, que, en este caso, siguiendo con la cámara de 150º, es 25.8º.
3. La distancia d tiene la misma fórmula que el ejemplo anterior.

$$d = vel_{lineal} * t_{acción} \quad (9)$$

4. Se calcula el valor z, que es la hipotenusa de un triángulo rectángulo.

$$z = \frac{d}{cos\ cos\ \mu} \quad (10)$$

5. De esta forma, como se tiene que los ambos rayos, s y $(s + z)$, son paralelos ya que son los rayos emitidos por el mismo haz, el del *scan_data[13]*, se tiene que el valor artificial calculado para este valor es la suma del *scan_data_past[13]* más el valor z calculado.

$$scan\ data[13] = scan\ datapast[13] + z \quad (11)$$

Este proceso se repetiría igual para el resto de las posiciones 10, 11 y 12. Para el caso de que el robot esté desplazándose hacia atrás, acción 0, que como se ha visto, solo sucede al principio del entrenamiento y cuando ha colisionado, el proceso de cálculo para estos cuatro valores sería similar. Centrándonos en la Figura 29, ahora el robot inicial sería el de arriba t_1 y después se desplazaría hacia detrás hasta t_0, entonces el *scan_data_past[13]* el haz de la derecha, el más largo, y habría que obtener el *scan_data[13]*, que sería el haz corto de la izquierda. El proceso es el mismo salvo que ahora se resta la z al *scan_data_past[13]*:

$$scan\ data[13] = scan\ datapast[13] - z \quad (12)$$

Por último, este proceso de relleno de los 4 valores traseros, solo se realiza cuando el movimiento es hacia delante o hacia atrás debido a que es posible calcular esta aproximación de las distancias, cuando el robot está realizando giros no se utiliza este método debido a que no es posible determinar esos valores Una vez analizado esto, es importante volver a destacar el proceso de relleno es una aproximación de lo que sucede en el entorno del robot, para que se lleve a cabo con datos más cercanos a la realidad que como sería si todos los valores no reales del láser fueran un valor promedio o uno aleatorio, con estos cálculos se pretende que los valores se puedan aproximar lo máximo a las distancias reales que se encuentra el robot alrededor suyo cuando se mueve.

4.6.2. Cálculos matemáticos en acciones de giro

Por último, se tienen los cálculos matemáticos para los casos que el robot está girando, es decir, que tiene velocidad lineal y angular. En estos casos, el proceso es similar al de cuando se repite la acción 2, pero en este caso es más complejo debido a que hora el agente se está desplazando con un movimiento circular uniforme (MCU) y la obtención del ángulo α no es tan directa como en el caso de ir en recto. En la Figura 30 se muestra un ejemplo de este caso, en el que el robot hace un giro hacia la derecha, ya que estaría haciendo la acción 3 con una velocidad lineal y angular determinada. Por su parte, en la Figura 31 se muestra más en profundidad los cálculos que se realizan.

Figura 30 Cálculos matemáticos para el relleno de datos en las acciones curvilíneas.

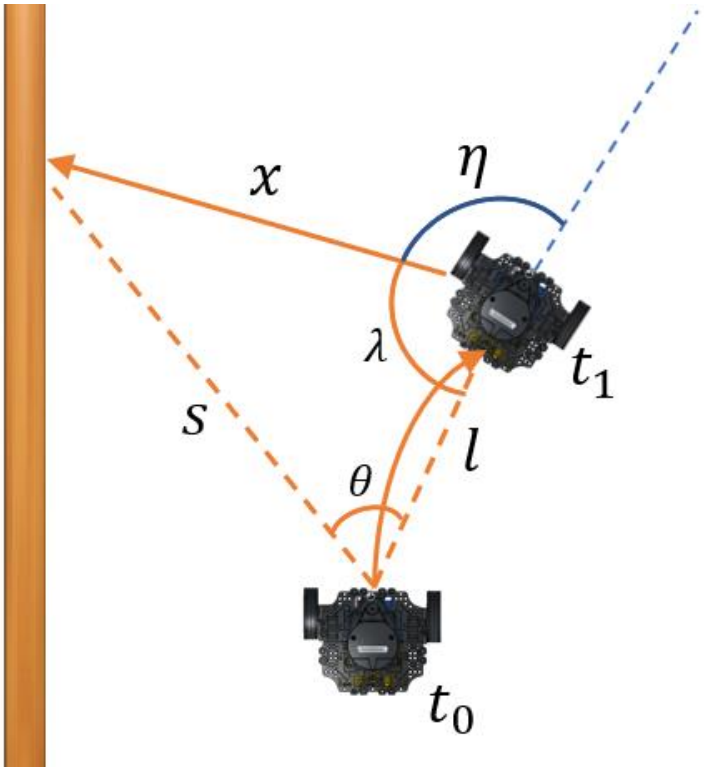
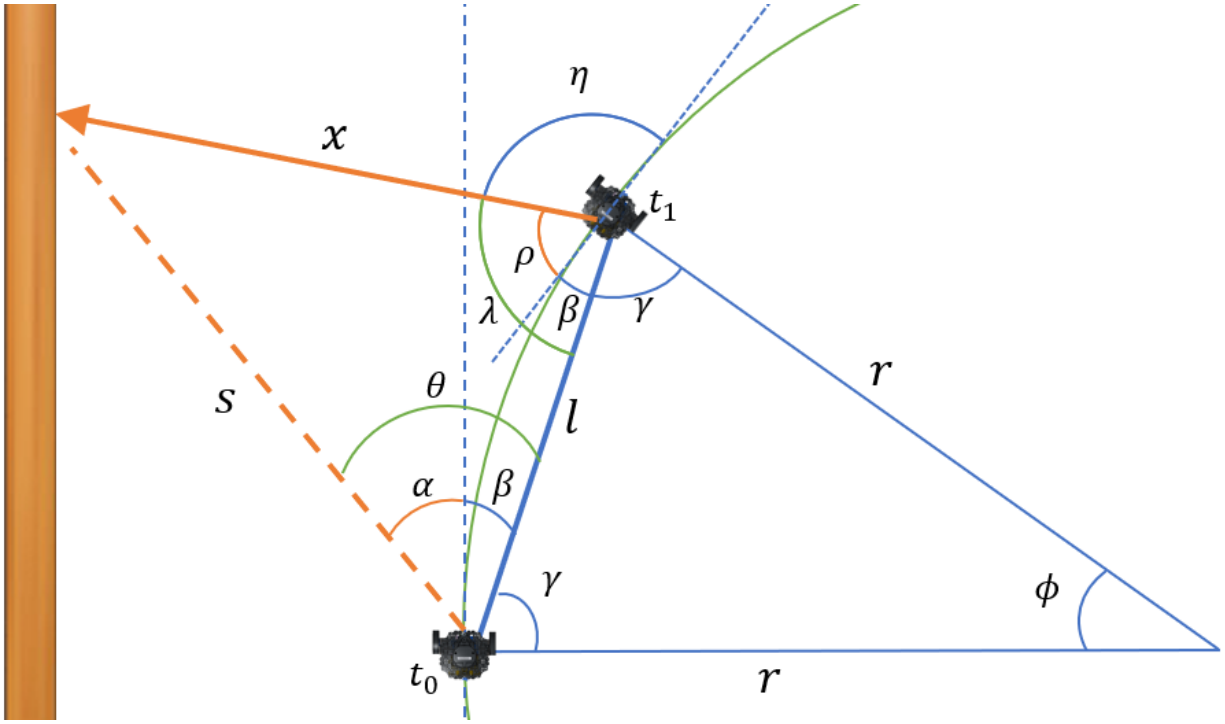


Figura 31 Descripción más detallada de los cálculos matemáticos para el relleno de datos en las acciones curvilíneas.



Se va a seguir con el ejemplo de la cámara de 150º, siendo los cálculos matemáticos realizados para este caso los siguientes:

1. En primer lugar, estos cálculos se van a realizar con todos los valores reales que otorgue la cámara del lado contrario al giro, es decir, como está girando hacia la derecha, los valores reales que se van a utilizar son los de la parte izquierda, del 0 al 5. En este caso se va a volver a poner el ejemplo para el sexto haz, por lo que la s es el *scan_data_past[5]* en t_0. Por su parte, el ángulo α es el del haz del *scan_data_past[5]*, 68.2º, pero ahora, a diferencia del caso de acciones lineal, se necesita un ángulo más β, para completar el ángulo que forma s con l, para poder calcular el valor artificial x.
2. Primero se va a calcular el valor l, que es la distancia en línea recta entre la posición inicial del robot en t_0 y la posición final del robot en t_1 después de realizar la acción y efectuar un MCU. Para calcular la l se necesita el radio de la circunferencia del MCU, que se calcula a partir de la velocidad lineal (v) y angular (w) del robot.

$$radio = \frac{v}{w} \qquad (13)$$

3. El siguiente paso es calcular el desplazamiento angular, ϕ, que realiza el robot entre la posición en t_0 y t_1, utilizando el tiempo transcurrido entre ambos momentos.

$$\phi = w * t_{acción} \qquad (14)$$

4. Con esto se tiene un triángulo isósceles entre las dos posiciones del robot y el centro de la circunferencia, por lo que a partir del radio y el ángulo ϕ, se obtiene la distancia l.

$$l = \sqrt{r^2 + r^2 - (2 * r^2 * cos\ cos\ \phi)} \qquad (15)$$

5. A partir del ángulo ϕ se puede obtener el ángulo γ.

$$\gamma = (\pi - \phi)/2 \qquad (16)$$

6. Y con el ángulo γ, se obtiene el ángulo β, que es su complementario.

$$\beta = \frac{\pi}{2} - \gamma \quad (17)$$

7. De esta forma, ya se obtiene el ángulo θ, que junto con s y l, permite calcular el valor artificial x.

$$\theta = \alpha + \beta \quad (18)$$

$$x = \sqrt{s^2 + l^2 - (2 * s * l * cos\ cos\ \theta)} \quad (19)$$

8. Los siguientes pasos son similares a los vistos con el movimiento lineal, que es determinar cuál es el haz al que corresponde el valor artificial x calculado. Para ello se obtienen el ángulo λ, y con el se calcula ρ, restando λ menos β , que es el ángulo complementario de γ.

$$\lambda = \left(\frac{l^2 + x^2 - s^2}{2 * l * x}\right) \quad (20)$$

$$\rho = \lambda - \beta \quad (21)$$

9. De esta forma se obtiene ρ, y su ángulo suplementario es η, que es el ángulo del haz que le corresponde el valor artificial x. Al igual que con el ejemplo lineal, se busca de todos los haces cual es el que está más cerca del ángulo η obtenido, para acabar el ejemplo, se va a suponer que se obtiene que el η es el de *scan_data[7]*, por lo que este tendrá el valor artificial de x, y habrá sido calculado a partir del *scan_data_past[5]*.

$$\eta = \pi - \rho \quad (22)$$

De esta forma es como se realiza el relleno de datos en los movimientos circulares, como antes, este proceso se realizaría para todos los valores reales del lado izquierdo, y en caso de ser el giro hacia la izquierda, serían los valores reales de la parte derecha los que se utilizarían para calcular los valores artificiales. Por último, es importante destacar que tanto para estos cálculos de las acciones de giro como para el de acción lineal, cuando se realiza un paso y se calcula un valor artificial, en el siguiente paso ese valor artificial se utiliza también para calcular otro valor artificial. Siguiendo el ejemplo anterior, se han analizado los valores reales de las posiciones 0, 1, 2, 3, 4 y 5, y con uno de ellos se ha conseguido el valor artificial de la posición 7, por lo que en el siguiente paso, para calcular los valores artificiales se utilizarán los valores del *scan_data_past* de las posiciones reales 0, 1, 2, 3, 4 y 5, como siempre, pero también se utilizará el valor artificial de la posición 7, ya que en el paso anterior se ha calculado su valor real, por lo que puede ser utilizado para calcular otro valor artificial.

Con esto queda explicado todo el tratamiento de los datos, en el que se ha visto los diferentes tipos de datos que había en función del sensor, como se obtenían, de qué forma se transformaban, filtraban y sincronizaban, y como finalmente se rellenan los valores en los casos que es necesario, con el fin de que los datos proporcionados a las redes neuronales sean lo más próximos a la realidad posible, para el que funcionamiento del agente sea óptimo. En la siguiente sección se expondrá las pruebas realizadas con los distintos sensores y se analizará los resultados obtenidos.

Capítulo 5. Pruebas y resultados

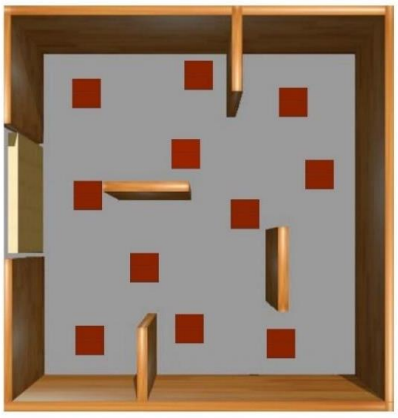
Para analizar el funcionamiento del robot en función del sensor utilizado, son diversas las pruebas que se han llevado a cabo. En ellas se pretende analizar la eficiencia de políticas de comportamientos obtenidas por una misma arquitectura de red neuronal y usando como datos de entrenamiento los obtenidos por diferentes sensores. Se usará el láser como sensor de referencia, al ser este el de mayor precisión y se comparará con los datos obtenidos por cámaras con diversas configuraciones de ángulo de visión. De esta forma se podrá analizar si el proceso de tratamiento, homogeneización, sincronización y relleno de valores ha sido adecuado y consigue que la funcionalidad de planificación de ruta que ejecuta el robot funcione correctamente independientemente del sensor que lleve incorporado.

Para realizar este análisis, ha sido necesario en primer lugar realizar 6 entrenamientos distintos, teniendo el robot en cada uno de ellos, una configuración del sensor distinta, de esta manera se han obtenido múltiples modelos con los pesos generados en el entrenamiento de cada configuración, ya que, a partir de esos modelos generados, se realizarán después las pruebas. Los entrenamientos de cada configuración han tenido una duración aproximada de 9 horas en las que el robot ha ido generando una política de comportamiento conforme iba aprendiendo del entorno. En esas nueve horas, han sido diversos los objetivos a lo largo de la habitación los que ha tenido que alcanzar el robot, apareciendo estos de forma aleatoria, alcanzando más o menos objetivos en cada episodio, dependiendo de la dificultad del mismo y la distancia del robot hasta él. Cada episodio dura hasta que el robot colisiona o hasta que el robot realiza 500 pasos o acciones sin colisionar. Antes de pasar con las pruebas y sus resultados, es necesario conocer cómo va a ser el entorno por el que se van a desplazar los agentes en busca de las metas delimitadas en las distintas pruebas.

5.1. Entorno del agente

El entorno de acción del agente está formado por una habitación que presenta obstáculos fijos en su interior. La forma de esta es prácticamente cuadrada, con unas medidas de 5.6 metros por 5.3 metros. En cuento a la altura de las paredes y obstáculos, estos son los suficientemente altos para que puedan ser detectados por los distintos sensores, en caso de estuviesen situados a distinta altura. En cuanto a los objetivos, se pretende que sean suficientes y estén completamente repartidos a lo largo del espacio, de forma que después del análisis de los resultados se pueda determinar si el agente, con cada uno de los distintos tipos de sensores y configuraciones, es capaz de llegar a todos los lugares de la habitación. En la Figura 32 se puede apreciar la distribución de la habitación utilizada, así como los 11 objetivos que tiene distribuidos.

Figura 32 Habitación con los objetivos donde se van a realizar los entrenamientos y pruebas.

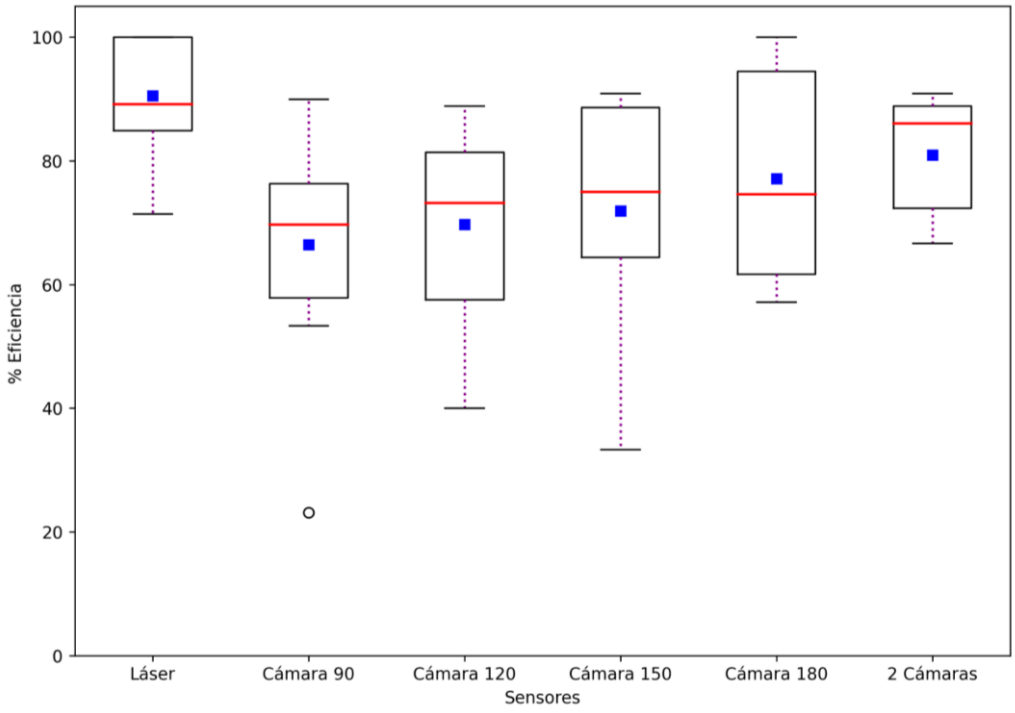


5.2 Sensores cargando un modelo propio

El primero de los análisis sobre las pruebas realizadas para comprobar el funcionamiento del robot con los distintos sensores, ha sido sobre el entrenamiento realizado por cada uno de ellos. Como se ha comentado, el robot ha sido entrenado en la habitación con los 6 tipos de sensores que se ha visto, con lo que ha conseguido generar multitud de modelos con los pesos generados conforme el robot iba aprendiendo del entorno con el paso de las acciones. Una vez generado los distintos modelos que contienen los pesos que definen la política de comportamiento, se ha seleccionado el último modelo de cada una de las configuraciones para ejecutarla y comprobar cómo funciona el robot con la política generada después de 10 horas de entrenamiento.

La Gráfica 1 contiene los resultados de las pruebas que se han hecho con cada sensor, y estas consistían en ejecutar el algoritmo de planificación del robot usando el último modelo del entrenamiento de cada sensor y ejecutando el mismo durante 5 minutos. Con los datos obtenidos se analizará la eficiencia de la política obtenida en términos de colisiones y objetivos alcanzados. Este proceso se ha repetido 10 veces, por lo que con cada sensor se han realizado 10 pruebas de 5 minutos. Los resultados numéricos obtenidos de las pruebas realizadas se pueden observar en el Anexo I. Datos numéricos.

Gráfica 1 Resultados de cada una de las configuraciones cargando un modelo propio y siendo ejecutado 10 veces durante 5 minutos.



Como se acaba de comentar, la Gráfica 1 consiste en un diagrama de caja-bigotes en la que se puede apreciar los resultados obtenidos por cada sensor. En este tipo de diagramas dan información sobre los resultados a partil de sus cuartiles; el primer cuartil indica que el 25% de los valores son menores o iguales al valor que marca, el segundo cuartil o mediana indica que el 50% de los valores son menores o igual al valor que representa y el tercer cuartil indica que el 75% de los valores son menores o igual que el valor que indica. Por su parte, con este diagrama también se aprecian los bigotes, que determinan el límite para identificar los valores atípicos, es decir, cualquier valor que sea mayor al bigote superior o menor al bigote inferior se considera un valor atípico. Los bigotes tienen un valor de 1.5 veces el Rango Intercuartílico (IQR), que es la diferencia entre el valor del tercer cuartil y el primero.

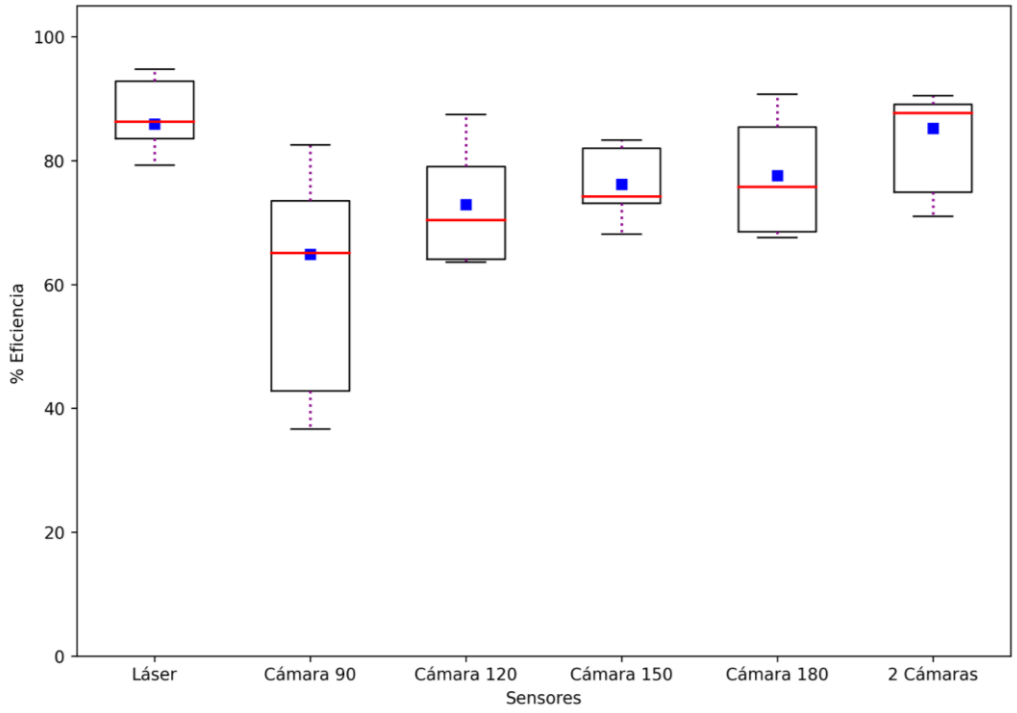
Analizando los resultados, en primer lugar, se puede observar que el sensor que mejores resultados ha obtenido ha sido el láser, como era esperado, con una media de logros de consecución de los objetivos en las 10 pruebas del 90.48%. Esto debido a que es el sensor más preciso y que no necesita de conversión de datos como el caso de las cámaras, además de que sus 24 valores son obtenidos directamente por el sensor y no son calculados como en el caso de las configuraciones de una cámara. El segundo sensor que mejor ha funcionado ha sido el de dos cámaras, con una media del 80.95%, dado que es un sensor que proporciona los 24 valores directamente desde el sensor. Este es el primer de los sensores en los que ha sido necesario un proceso de conversión, filtro y sincronización, y como se puede observar los resultados son muy buenos.

Por parte de las configuraciones de una sola cámara, se aprecia la evolución ascendente del porcentaje de logros conforme el sensor aporta más datos reales. La configuración que menos objetivos alcanza es la de 90º. Debido a que es la que menos datos reales aporta directamente (8 valores) y además es necesario realizar el proceso completo de conversión, filtro, sincronización y relleno de datos. De todas formas, es destacable que solo aportando 8 de los 24 valores, se consiguen unos resultados del 66.13% de acierto, lo que indica que el proceso de relleno de datos es positivo y se consigue que el robot tenga un funcionamiento bueno. En cuanto al resto de configuraciones, los resultados mejoran, obteniéndose con la cámara de 120º una media de 69.72%, con la de 150º 71.92% de logros y finalmente con la cámara que más datos reales aporta, la de 180º, se consiguen unos resultados muy buenos con un acierto del 77.08%, no tan lejos de los resultados de las 2 cámaras que sí disponen de todos los valores reales.

Un aspecto que se puede apreciar en la Gráfica 1 es la variabilidad de los resultados en función de el sensor, ya que a menor amplitud del rango intercuartílico y de los límites de la caja de bigotes, menor variabilidad existirá en los resultados. Se puede apreciar que tanto el láser como las 2 cámaras tienen una caja compacta y los bigotes cortos, especialmente el sensor de 2 cámaras, lo que nos indican que los resultados en ambos sensores son más similares y no presentan mucha disparidad en las diferentes pruebas. Para el caso de las configuraciones de una sola cámara se aprecia más variabilidad en los resultados, ya que las cajas son más alargadas como es el caso de la cámara de 180º. En el caso de la cámara de 120º y 150º presentan bigotes largos, y el caso de la cámara de 90º presenta valores atípicos, con una forma de caja compacta y bigotes no muy excesivos, además presenta un valor atípico debido a que en la prueba 8, los resultados fueron de un porcentaje de éxito del 23.08%. Este valor atípico con baja tasa de eficiencia se puede explicar tomando en cuenta que todos los objetivos a alcanzar son aleatorios, por lo que durante esa prueba los objetivos más complicados de alcanzar pudieron ser escogidos varias veces.

Una vez analizado los primeros resultados se puede afirmar que los resultados son lógicos, ya que los sensores que mejores resultados han obtenido han sido los dos que aportan los 24 valores reales, en especial el láser, que es el más preciso, mientras que las configuraciones de una sola cámara aportan buenos resultados, aunque con unos porcentajes de éxito menores que las dos configuraciones comentadas, demostrando que aunque solo se cuente con 8 valores reales y se estimen los restantes 16 valores se puede obtener una eficiencia muy cercana a la ofrecida por el láser. La siguiente prueba que se ha realiza es similar a esta, ya que se vuelve a cargar en cada sensor un modelo de su propio entrenamiento, pero en este caso las pruebas han sido más largas de 25 minutos. De esta forma, la Gráfica 2 muestra los resultados de cada sensor en 5 pruebas de 25 minutos cada una, estando los resultados utilizados para la gráfica en Resultados Gráfica 2.

Gráfica 2 Resultados de cada una de las configuraciones cargando un modelo propio y siendo ejecutado 5 veces durante 25 minutos.



Antes que nada, se puede observar a simple vista, que en general los resultados presentan menos variabilidad ya que los bigotes de las distintas cajas son considerablemente más cortos y en algunos casos casi inexistentes. Esto es lógico, debido a que al ser pruebas más largas, la probabilidad de que solo salgan los objetivos más complicados en menor, y lo normal es que, en 25 minutos, el número de veces que sale cada objetivo sea más similar, por eso los resultados hay menos disparidad entre los resultados de cada prueba. Además, a medida el robot recolecta datos también va ajustando el modelo cargado al nuevo sensor. En cuanto a la media de logros se aprecia como el orden de que sensor obtiene mejores resultados es el mismo que para la primera prueba, siendo el láser el mejor de todos, consiguiendo un 85.92% de los objetivos, aunque es está prueba está muy cerca las 2 cámaras, que tiene un éxito del 85.24%. Por parte de las configuraciones de una cámara, la de 90º es la que menor media presenta con un 64.87% y la que mayor variabilidad presenta, mientras que conforme aumenta el ángulo de visión la cámara, mejores son los resultados una vez más.

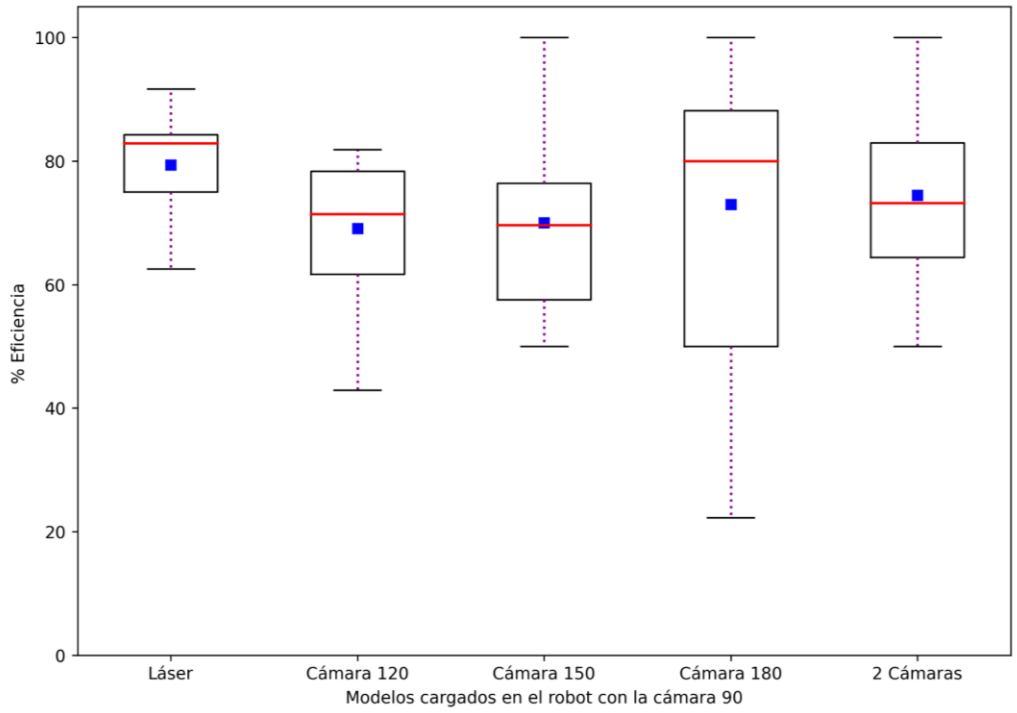
Una vez más, son la configuración del láser y 2 cámaras, junto con la de 150º en este caso, las que menos dispersión encuentran en sus resultados, teniendo las cajas muy compactas. En estas pruebas no se aprecian para ninguna de las configuraciones valores atípicos, lo que es normal, ya que como se ha comentado, en pruebas más largas es más normal que los resultados sean más parecidos que pruebas más cortas. Una vez analizados estas dos pruebas en las que se carga un modelo propio de cada configuración, se puede concluir que en aquellas que ha sido necesario realizar un tratamiento de los datos, es decir, en todas menos el láser, los resultados han sido positivos. Consiguiendo un porcentaje de éxitos buenos a pesar de disponer de muy pocos datos reales, como es el caso de la cámara de 90º, así como consiguiendo resultados similares a los que se obtienen con el láser para el caso de las 2 cámaras.

5.3 Pruebas usando una cámara de 90º y el modelo obtenido por el resto de configuraciones

Este segundo grupo de pruebas consiste en usar el robot con una cámara de 90 y cargar el último modelo del resto de configuraciones. Se realizan las mismas pruebas mencionadas anteriormente, primero se realizan 10 pruebas de 5 minutos, y después 5 pruebas de 25 minutos cada una. Lo que se pretende con estas pruebas es determinar la eficiencia del funcionamiento del robot con una cámara de 90º, cuando se carga un modelo que fue entrenado por sensores de mayor ángulo de visión. Esta prueba es interesante porque en las anteriores, los resultados que se obtenían de la cámara de 90º, eran resultados a partir de cargar un modelo propio que se había obtenido a partir de los datos que le ofrece la cámara de 90º, que son los 8 reales, más los valores promedio y los valores artificiales que se calculan. Entonces ahora, al cargar modelos que son mejores, ya que se han obtenido a partir de más datos reales, se aprecia como afectan los modelos y si al cargarlos, el robot con la cámara de 90º funciona mejor.

Por otra parte, con estos resultados se puede analizar cómo es la compatibilidad de modelos entre diferentes configuraciones, ya que uno de los objetivos de este trabajo era analizar el funcionamiento del robot cuando se le carga un modelo que no fue entrenado con sus datos, sino que procede del entrenamiento de otro robot con un sensor completamente distinto, como es el caso de las pruebas que se muestran gráficamente a continuación en la Gráfica 3, estando los resultados de cada prueba individual en Resultados Gráfica 3.

Gráfica 3 Resultados de cargar en la cámara de 90º modelos que han sido generados en los entrenamientos de las otras configuraciones en 10 pruebas de 5 minutos cada una.



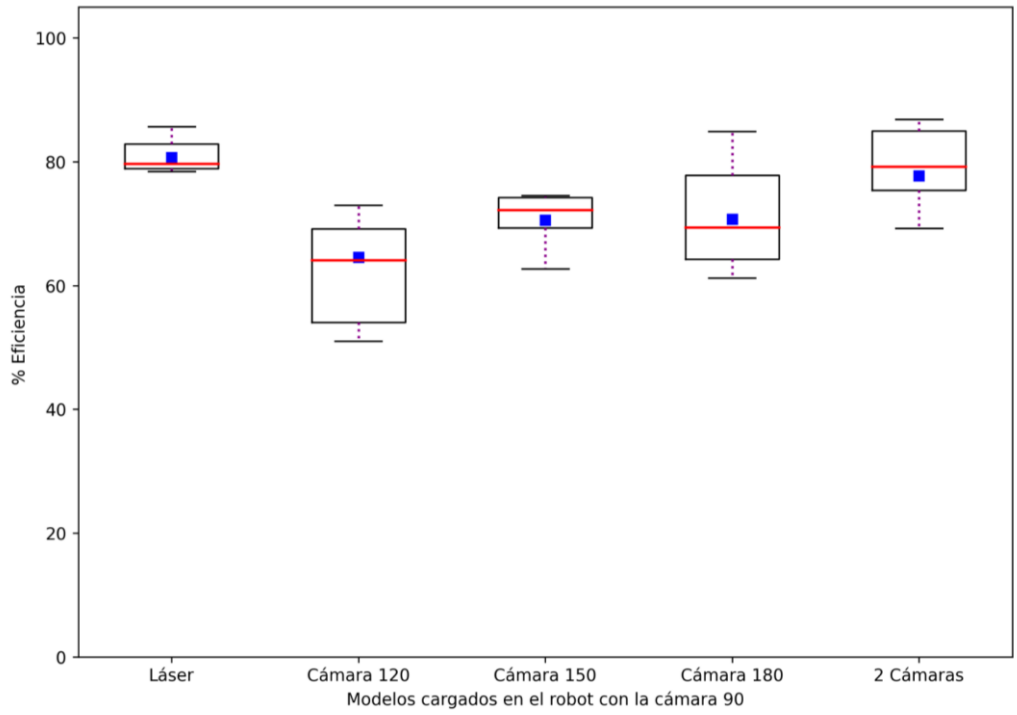
Como se aprecia en la Gráfica 3 los resultados que se obtienen al ejecutar el robot con la cámara de 90º son mejores a los que se obtenían cuando se cargaba un modelo propio de la cámara de 90º con es el caso de la Gráfica 1, en la que el porcentaje de logros obtenidos al cargar un modelo propio era del 66.13%. En este caso, cargando modelos de otras configuraciones los resultados siempre son mejores, lo que demuestra la influencia del entrenamiento en el funcionamiento posterior del robot, ya que aunque se esté ejecutando un robot con una cámara que solo da 8 valores reales, como los pesos que está utilizando para calcular las acciones se han obtenido a partir de datos con más valores reales, el funcionamiento del robot mejora considerablemente.

Apreciando las caja-bigotes de la Gráfica 3 se ve cómo como es con el modelo del entrenamiento con el láser con el sensor que mejores resultados se obtiene, y es bastante destacable la mejora que se produce al ejecutar la cámara de 90º con un modelo propio o con el del láser, ya que se pasa de un porcentaje de éxito del 66.13% a un 79.36%, demostrándose la importancia del entrenamiento y los datos que se utilizan en él. Una vez más con las 2 cámaras se obtienen los segundos mejores resultados, y es que cargando un modelo de esta configuración se obtiene que en las pruebas el robot con la cámara de 90º alcanza el 74.43% de los objetivos.

En cuanto a las configuraciones de una sola cámara, se vuelve a apreciar una evolución favorable conforme se aumenta el ángulo de visión de la cámara. Se ha obtenido después de las pruebas, un porcentaje de éxito del 69.05% de media cuando se carga un modelo del entrenamiento con la cámara de 120º, unos resultados del 69.99% con la cámara de 150º y finalmente, utilizando un modelo de la configuración de una cámara que mejor función del entrenamiento con la cámara de 120º,del entrenamiento con la cámara de 120º,del entrenamiento con la cámara de 120º,del entrenamiento con la cámara de 120º,a, la de 180º, se obtiene que se alcanzan el 72.97% de los objetivos, mejorando una vez más los resultados que obtiene el robot de 90º si carga un modelo propio.

En cuanto a la variabilidad de los resultados de cada prueba, se observa que el modelo más compacto en resultados son el del láser. Mientras que la configuración de una cámara de 150º y la de 180º presentan mayor diferencia en sus resultados ya que su RIC es el más amplio de todos. A diferencia de la Gráfica 1, que era también sobre pruebas cortas de 5 minutos, en esta no se aprecia ningún resultado anómalo que esté fuera del rango que abarca los bigotes de las cajas. A continuación, en la Gráfica 4 se muestran los resultados de las mismas pruebas de la gráfica anterior, cargando modelos de los diferentes sensores en el robot con la cámara de 90º, pero en este caso las pruebas han sido 5 con una duración de 5 minutos.

Gráfica 4 Resultados de cargar en la cámara de 90º modelos que han sido generados en los entrenamientos de las otras configuraciones en 5 pruebas de 25 minutos cada una.



Una vez más a primera vista se aprecia un menor tamaño de las cajas, debido a que al ser más largas las pruebas, más complicado es que se den resultados anómalos, sumado a que en este caso hay la mitad

de resultados en comparación con la gráfica anterior, por cada sensor 5 pruebas en vez de 10. En este caso los resultados obtenidos son muy similares a los obtenidos en la Gráfica 3 con las 10 pruebas de 5 minutos, el robot de 90º con el modelo del láser cargado tiene unos resultados muy buenos, del 80.65%, muy superiores a los que se obtenían en la Gráfica 2 cuando se cargaba en el robot con la cámara de 90º un modelo de él mismo, siendo en ese caso el porcentaje de éxito del 64.87%. En cuanto a cargar el modelo entrenado de la configuración con 2 cámaras, se obtienen mejores resultados que en las pruebas de 10 minutos, consiguiendo en este caso una tasa de éxito del 77.73%, confirmando de nuevo que los resultados obtenidos en el entrenamiento de esta configuración han sido muy positivos, consiguiendo resultados muy parejos al sensor de referencia, el láser.

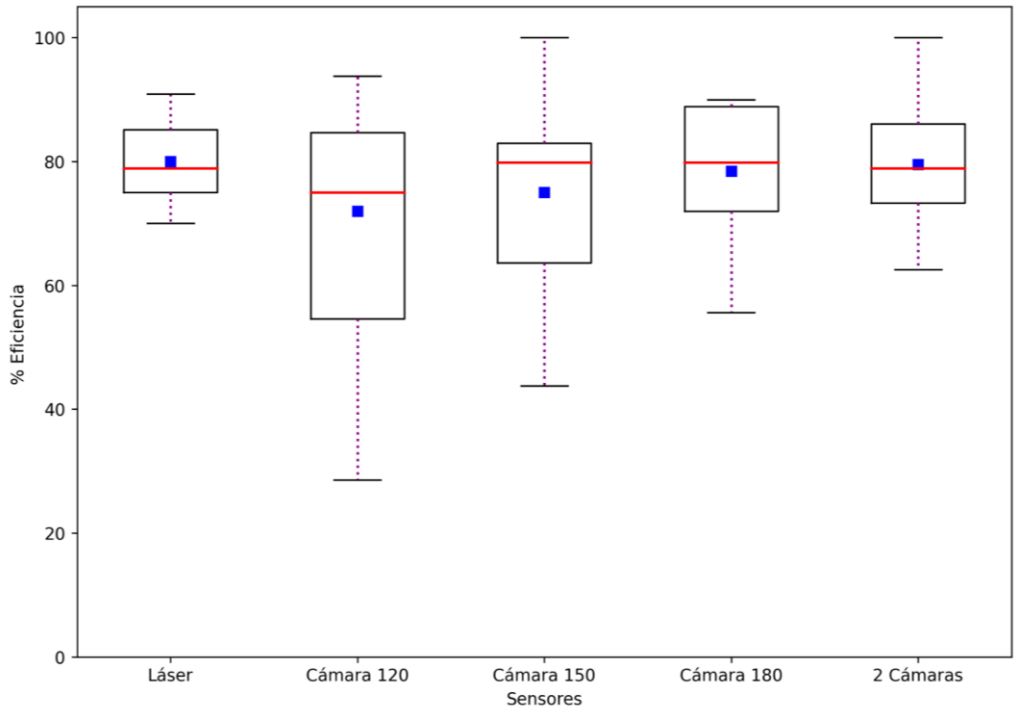
En cuanto a las configuraciones de una sola cámara, es destacable que los resultados de cuando se carga el modelo de la cámara de 120º son peores en comparación con las pruebas de 5 minutos, pero a pesar de ello se obtiene un buen valor de éxito del 64.53% prácticamente igual a los resultados que obtiene el robot con cámara de 90º grados cuando lleva un modelo del entrenamiento de esa misma cámara. En cuanto a las dos configuraciones restantes, cargando el modelo de la cámara de 150º y de 80º, con ambas se obtienen unos resultados muy similares, del 70.59% y 70.68%.

A modo de conclusión para este segundo grupo de pruebas, se puede comprobar claramente cómo utilizar modelos mejor entrenados, aunque sean de otro sensor, hace que el funcionamiento mejore considerablemente. El robot utilizando la cámara de 90º, como se ha visto en las dos primeras pruebas, era el que peor funcionaba teniendo la tasa de éxitos más baja de todos, tanto en las pruebas de 5 minutos como las de 10, pero cuando en vez de cargarle su propio modelo entrenado, se le ha cargado modelos del resto de configuraciones, que funcionaban mejor, el porcentaje de aciertos ha aumentado considerablemente en algunos casos, como son los casos de los modelos del láser y de las dos cámaras. Demostrando que robot puede funcionar con un modelo de entrenamiento que no ha sido realizado por él mismo, y que además si ese modelo es bueno, hará que el robot alcance una parte importante de los objetivos independientemente que el sensor que utilice el robot no sea el mejor o el que más datos reales aporte.

5.4 Pruebas en las configuraciones cargando un modelo de la cámara de 90º

Por último, está el tercer grupo de resultados, que se han obtenido a partir de unas pruebas en las que se ha seguido la metodología contraria al segundo grupo de pruebas, en las que se cargaban los modelos de cada una de las configuraciones en el robot, utilizando este el sensor con menor ángulo de visión, la cámara de 90º. En este caso se va a cargar un modelo obtenido en el entrenamiento de la cámara de 90º en el resto de configuraciones, de forma que se va a ejecutar el robot con el láser, con las dos cámaras y con la cámara de 120º, 150º y 180º, pero cargando en ellos un modelo obtenido con la cámara de 90º. De esta forma se pretende analizar el funcionamiento del robot con cada una de las configuraciones cuando este esté utilizando un modelo que no es el propio de esa configuración, y que procede de la configuración que menos datos reales aporta, es decir, que es la que tiene menor porcentaje de éxito. En Resultados Gráfica 5 se encuentran los resultados individuales de cada una de las pruebas que se muestran en la Gráfica 5.

Gráfica 5 Resultados de cada configuración cargando un modelo de la cámara de 90º en 10 pruebas de 5 minutos cada una.



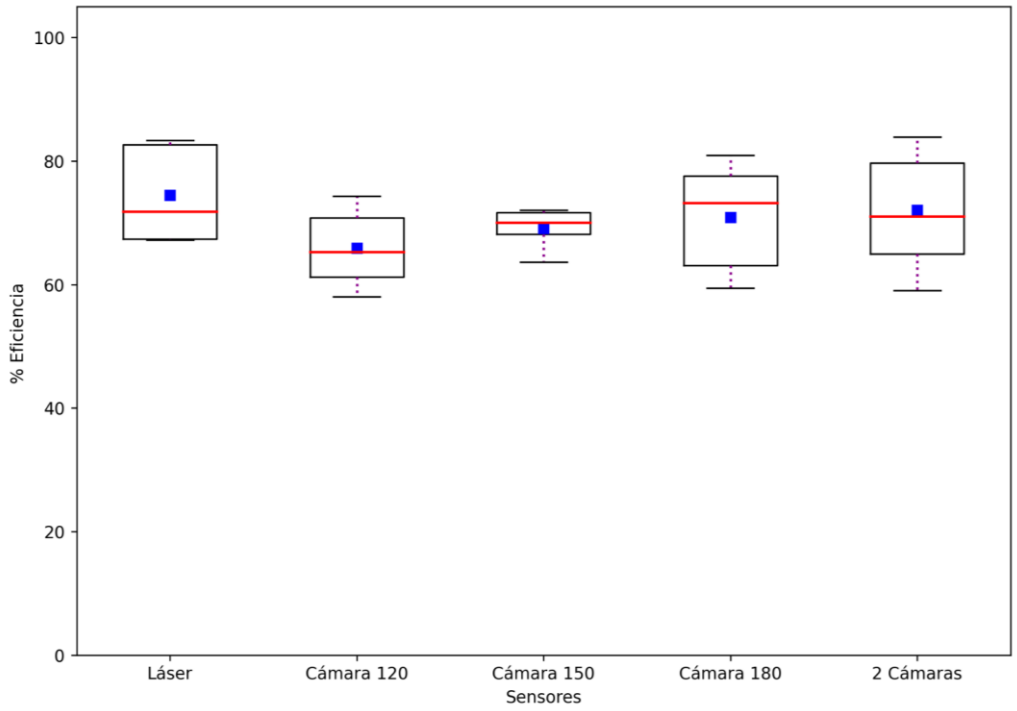
Lo primero que se puede apreciar en la Gráfica 5 es que los resultados, entre cada configuración, son más similares que en el resto de pruebas, ya que todo de la configuración con peores resultados, la cámara de 120º, y la mejor, el láser, hay menos de un 10% de diferencia en el porcentaje de objetivos alcanzados. Por otra parte, una vez más se mantiene que contra más datos reales aporta el sensor, mejores son los resultados que se obtienen, ya que la segunda mejor configuración es la de las dos cámaras y después están las de una sola, siendo la cámara de 180º grados la que mejores resultados ofrece en comparación con la de 150º y 120º.

Lo más interesante de esta prueba es utilizar un modelo de menor tasa de eficiencia en el resto de configuraciones, es poder comparar los resultados que se obtienen cuando se está cargando un modelo propio, o uno de un sensor con menor ángulo de visión. Para realizar esta comparación hay que observar los resultados obtenidos en la Gráfica 1, donde cada configuración carga un modelo propio en 10 entrenamientos de 5 minutos. Observando esos resultados y los obtenidos ahora, se aprecia que el láser funciona considerablemente mejor con su modelo, ya que con él tenía un porcentaje de éxito del 90.48% mientras que con el modelo de la cámara de 90º, es del 80.02%. Esto es lo lógico debido a que el modelo del láser está mejor entrenado al disponer de más datos reales para el entrenamiento.

En cambio, con el resto de configuraciones no sucede exactamente lo mismo, debido a que tanto con sus propios modelos como con el modelo del láser se obtienen resultados similares. Una de las posibles causas de que los resultados con el modelo de la cámara de 90º sean parecidos a cuando cargan un modelo propio, cuando con este último deberían ser en teoría mejores, es el hecho ya comentado de la duración de las pruebas, ya que al ser de 5 minutos es posible que salgan los objetivos más sencillos solo o los más complicados, haciendo que se desvirtúen los resultados.

Por este motivo es interesante realizar las pruebas de 25 minutos también para comparar los resultados, y ver en este caso, si se repite el patrón con algunas de las configuraciones de que sus resultados sean parecidos cargando su propio modelo o el de la peor configuración. En la Gráfica 6 se aprecian estos resultados gráficamente de las 5 pruebas de 25 minutos cada una, en cada configuración con el modelo del entrenamiento del robot con la cámara de 90º, mientras que en la tabla de Resultados Gráfica 6 se pueden observar los resultados de cada una de las pruebas individuales.

Gráfica 6 Resultados de cada configuración cargando un modelo de la cámara de 90º en 5 pruebas de 25 minutos cada una.



Si se analizan los resultados de la Gráfica 6 y se comparan con los de la Gráfica 2, que son las pruebas de 25 minutos con cada configuración cargando su modelo, se puede comprobar que cada una de las configuraciones funciona mejor, consiguiendo un mayor porcentaje de objetivos, cuando utiliza sus propios modelos del entrenamiento, siendo en algunos casos una diferencia importante, como el caso del láser o la configuración de 2 cámaras, en que la diferencia de éxito entre cargar el modelo de la cámara de 90º o cargar su propio modelo es de más del 10%, lo cual es completamente lógico, ya que estas dos configuraciones han creado sus modelos a partir de los 24 valores de distancia reales, mientras que el de la cámara de 90º solo dispone de 8 más los rellenados matemáticamente.

De todas formas, a pesar de que no funcionen las configuraciones tan bien como cuando cargan su propio entrenamiento, los resultados cargando la configuración con menor tasa de eficiencia, son bastante bueno, el láser, por ejemplo, consigue un 74.48% de consecución de objetivos y la configuración de 2cámaras un 72.05%, cercano al 70.88% del sensor de una cámara de 180º. Todo esto demuestra que si un robot con un buen sensor entra en una habitación donde los pesos que se utilizan han sido generados a partir del entrenamiento de un sensor de menor tecnología, el funcionamiento del robot será adecuado y podrá alcanzar un porcentaje elevado de los objetivos, a pesar de que los pesos utilizados no sean los más idóneos.

Capítulo 6. Conclusiones, limitaciones y líneas futuras

En cuanto a las conclusiones que se extraen al finalizar este trabajo de fin de grado, cabe resaltar que se han cumplido con creces los objetivos planteados al principio de este. En primer lugar, se ha conseguido conocer y entender en gran parte los algoritmos de IA y en especial los de RL a través de su programación en Python, lenguaje que se ha aprendido a programar gracias a la realización del trabajo. El aprendizaje de estos aspectos no solo era necesario para poder llevar a cabo este trabajo, sino que además ha sido realmente importante pensando en el futuro, debido a que son tecnologías muy presentes en la actualidad y es muy positivo su aprendizaje. Por su parte, sucede lo mismo con el aprendizaje de ROS, que se ha conseguido conocer y entender su funcionamiento, así como la programación en ese entorno.

En cuanto al trabajo realizado, el objetivo principal era adaptar los datos procedentes de diferentes sensores para convertir los datos heterogéneos que proporcionaban a un formato común que para que pudieran ser utilizados en un algoritmo de planificación de rutas mediante RL que requiere de datos homogéneos, es por ello que lo primero que ha sido necesario ha sido conocer qué tipo de datos proporcionaban los sensores para saber de qué forma adaptarlos para que todos los sensores puedan ser utilizados en el algoritmo. Para ello se ha establecido que se iban a utilizar dos tipos de instrumentos de medida, el láser y la cámara. El primero de ellos, por su precisión y estructura de datos, se estableció como sensor de referencia, por lo que los datos de la cámara iban a ser los que debían ser adaptados a la estructura de los del láser.

En cuanto a la cámara, cinco han sido las distintas configuraciones que se han desarrollado, una de ella formada por dos cámaras de 180º, una frontal y otra trasera, y el resto de configuraciones han consistido en una única cámara con diferentes ángulos de visión, 90º, 120º, 150º y 180º. El tratamiento de los datos procedentes de estas 5 configuraciones ha sido la parte central y principal de este trabaja de fin de grado, debido a que los datos proporcionados por estos sensores presentaban una estructura y contenido muy distinto al del láser.

El primer paso del tratamiento de los datos de la cámara ha sido el proceso de conversión, ya que como se ha visto, la información que aporta este tipo de sensor es una nube de puntos 3D que ROS proporciona a través de mensajes del tipo *PointCloud2*, mientras que con el láser se obtiene un escaneo 2D a través de los mensajes ROS con del tipo *LaserScan*, por lo que ha sido necesario convertir los mensajes de la cámara al tipo *LaserScan*, y eso ha sido posible gracias a un paquete de la librería de ROS llamado *pointcloud_to_laserscan*. De esta forma, aplicando un nodo ROS con este paquete se ha conseguido el primer objetivo de obtener los datos de distancia de todos los sensores y configuraciones con el mismo tipo de mensajes.

El siguiente paso a realizar era el de sincronización los sensores, esto era necesario realizarlo por dos situaciones; la primera para la configuración de las dos cámaras, debido a que era necesario que las 24 medidas de distancia, 12 de cada cámara, se hubiesen tomado justo en el mismo instante, por lo que ambas cámaras debían estar sincronizadas para tomar las medidas siempre al mismo tiempo. El segundo caso es para las configuraciones de una sola cámara, ya que, aunque los datos utilizados para el entrenamiento y decisión de las acciones eran los de la cámara, se necesitaban los datos del láser para realizar una comprobación de los resultados, por lo que, en las configuraciones de una sola cámara se sincronizaban esta con el láser.

Para llevar a cabo esta sincronización se ha utilizado un paquete de ROS llamado *message_filters* que contiene dos tipos de filtros, *ApproximateTime* y *ApproximateTimeSynchronize*, con los que se filtran los datos de los diferentes sensores que se necesitan sincronizar y se crea un nuevo mensaje en el que están los datos sincronizados. Además, el proceso de sincronización también servía para ordenar los valores que se obtenían con la cámara, con el objetivo de que estuviesen dispuestos en el vector final

que se utiliza para entrenar las redes con el mismo orden de los haces que tiene el láser. Por lo que, en este segundo paso de tratamiento de los datos, se conseguía filtrarlos, sincronizarlos con el otro sensor y ordenar los valores.

Por último, en las configuraciones de una única cámara también era necesario realizar un proceso de relleno de datos, debido a que, con esas cuatro configuraciones, los valores de distancia que se obtenían eran menores a los 24 necesarios, ya que el ángulo de visión es menor a los 360º que tiene el láser o la configuración de las dos cámaras. El proceso de relleno ha sido la parte más laboriosa del trabajo debido a que había que desarrollar un sistema por el cual fuese posible rellenar valores que no aportaba la cámara y que además estos fueran precisos. Para conseguir esto se utilizó trigonometría para determinar los valores que no se obtenían por la cámara ya que estaban fuera del ángulo de visión de está, partir de los valores reales que sí se obtenían con la cámara.

Para ello se diferenció entre las acciones que solo tenían velocidad lineal y las que tenían tanto velocidad angular como linear y se realizaron los cálculos en cada uno de ellas, para que en cada acción se puedan rellenar los máximos valores posibles con valores calculados muy precisos. De esta manera se consiguió que, aunque las cámaras no aportasen los 24 valores directamente, a partir de rellenar los valores con cálculos matemáticos, se pudiesen completar todos los datos necesarios. Con todo esto se consiguió adecuar las configuraciones de las cámaras para que tuviesen un mismo formato que el láser y para aquellas que no tuviesen los datos suficientes, rellenarlos con valores adecuados que hagan que el funcionamiento del robot con ellas no sea óptimo.

Una vez realizados todos los objetivos sobre el tratamiento y adecuación de los datos, solo queda realizar una revisión del análisis de los resultados, que son los que determinan si el trabajo realizado ha sido adecuado y si se ha cumplido el objetivo principal del trabajo. Para determinar esto, se han realizado 3 grupos de pruebas, y en cada grupo, dos tipos diferentes; uno consistía en realizar 10 pruebas de 5 minutos con cada sensor y el otro tipo eran 5 pruebas de 25 minutos cada una, de esta forma se analiza el comportamiento del robot cuando se ejecuta poco tiempo y cuando se realizan pruebas más largas.

El primer grupo de pruebas se ha basado en cargar en cada configuración el último modelo generado en el entrenamiento realizado durante 9 horas. De esta manera se comprobaba el funcionamiento del robot con el modelo generado durante su entrenamiento, y así determinar que configuración funcionaba mejor y ver las diferencias entre el comportamiento de cada una. Los resultados de estas primeras pruebas han sido positivos porque ha confirmado la suposición inicial de que el láser es el sensor que mejor debería funcionar, es el de referencia, y contra mayor número de datos reales aporte el sensor, mejor debería ser el funcionamiento del mismo. Como se ha visto en los primeros resultados, el sensor que mejor funcionaba era el láser junto con las 2 cámaras, y después las configuraciones de una sola, la mejor era la de 180º.

Otro aspecto muy positivo a valorar en estas primeras pruebas ha sido que los sensores que menos datos reales aportan, las configuraciones de una sola cámara, aunque no han tenido resultados tan excelentes como el láser o las dos cámaras, como era de esperar, han obtenido un elevado porcentaje de éxito en la consecución de objetivos que hace ver que el proceso de tratamiento de los datos, con la conversión, sincronización y rellenos de estos, ha sido muy exitoso.

El siguiente grupo de pruebas iba dirigido a comprender que sucedía si se cargaba un modelo que no había sido entrenado por la propia configuración, en un robot que utiliza el sensor con el menor ángulo de visión que es la cámara de 90º. Los resultados obtenidos en este apartado también han sido muy positivos, porque se ha comprobado que cuando el robot con la cámara de 90º, carga un modelo muy bien entrenado, como el del láser o las dos cámaras, obtiene unos resultados muy buenos, mucho mejores que los que genera cuando carga un modelo de su entrenamiento con la cámara de 90º. Esto es

muy positivo porque, aunque se utilice un sensor no tan preciso, si el modelo que se utiliza es muy bueno, los resultados se verán notablemente mejorados.

Para finalizar, el último grupo de pruebas tenía la finalidad de comprobar que sucedía en cada configuración cuando se cargaba un modelo con la de la cámara de 90º. Lo que se ha obtenido en los resultados ha sido lo esperado, que es que los resultados que se obtienen son peores que si cada configuración cargase un modelo propio que es mejor que el modelo de la cámara de 90º. Además, el aspecto a valorar de esta prueba es que se ha comprobado que si un robot con un sensor muy bueno, si utiliza un modelo que no es tan bueno, aunque no obtenga unos resultados excelentes, sí que serán buenos y mejores que los que obtiene la cámara de 90º con ese mismo modelo propio.

En cuanto a las limitaciones del trabajo, la principal ha sido el ordenador utilizado, debido a que ha sido necesario cambiar hasta en dos ocasiones de ordenador para poder realizar las pruebas debido o bien este no disponía de núcleos suficientes para ejecutar el algoritmo con la paralelización o el ordenador no era lo suficientemente potente para realizar el entrenamiento de las redes neuronales al mismo tiempo que utilizaba la red para calcular las acciones, por lo que el robot ejecutaba estas de forma excesivamente lenta que hacía que el funcionamiento fuera inadecuado. Así mismo, como se comentó, el motivo para separar los procesos a realizar por el servidor en dos nubes distintas, es reducir las capacidades computacionales necesarias en el ordenador, ya que de ese modo se iba a realizar el entrenamiento y la toma de decisiones de una forma eficiente. Por este motivo, el ordenador utilizado para el proyecto ha sido una limitación principalmente por el tiempo que se ha perdido solucionando los problemas derivados de su uso.

Por último, son muchas las líneas futuras de trabajo con las que se puede mejorar y ampliar el alcance de este proyecto, desde ampliar los tipos de sensores con los que realizar las pruebas hasta poder implementar el trabajo de forma física. Para resumir, estas serían las principales líneas futuras a considerar:

- Realizar las mismas pruebas en otra habitación distinta con otra distribución, para analizar si el robot con los distintos sensores se comporta de la misma manera.
- Probar nuevas configuraciones de las cámaras, principalmente para la configuración en la que se utilizan dos de ellas, utilizando en vez de dos cámaras de 180º, dos de 90º o una combinación de cámaras con diferentes ángulos de visión, para comprobar si el funcionamiento sigue siendo adecuado.
- Realizar pruebas en las que el robot pasa de una habitación a la otra, utilizando los pesos del entrenamiento que hay cargados en cada una de ellas, comprobando si el robot se adapta a los pesos de la nueva habitación.
- Desarrollar un algoritmo de colaboración entre agentes de forma que se puedan lanzar dos agentes en una misma habitación, y que cada uno además de evitar los obstáculos estáticos como realizar ahora, puedan evitar también los dinámicos, que sería el otro agente.
- Llevar a cabo las pruebas en un ordenador más potente para que solo se requiera de dos núcleos para correr el algoritmo, uno para los procesos del robot y otro que realice los procesos que ahora se realizan en dos núcleos independientes, la nube 1 y 2. De esta forma se reducen las comunicaciones entre núcleos y se comprueba si esto mejora el funcionamiento del robot.
- Implementar el proyecto de forma física utilizando una red de 5G para la comunicación entre el agente y el servidor. De esta manera el agente sería un robot físico y habría un servidor realizando el entrenamiento de las redes y el cálculo de las acciones, y la comunicación entre ambos sería mediante una red de 5G.

Referencias

[1] I. J. González-Hernández, B. Armas-Alvarez, M. Coronel-Lazcano, N. Maldonado-López, O. Vergara-Martínez, y R. Granillo-Macías, «El desarrollo tecnológico en las revoluciones industriales», *Ingenio y Conciencia Boletín Científico de la Escuela Superior Ciudad Sahagún*, vol. 8, n.º 16, pp. 41-52, jul. 2021, doi: 10.29057/ESCS.V8I16.7118.

[2] F. Rozo-García, «Revisión de las tecnologías presentes en la industria 4.0», *Revista UIS Ingenierías*, vol. 19, n.º 2, pp. 177-191, may 2020, doi: 10.18273/REVUIN.V19N2-2020019.

[3] R. Siegwart y I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, 1.ª ed. Massachusetts: MIT Press, 2004. Accedido: 7 de febrero de 2023. [En línea]. Disponible en: https://www.ucg.ac.me/skladiste/blog_13268/objava_56689/fajlovi/Introduction%20to%20Autonomous%20Mobile%20Robots%20book.pdf

[4] S. Hakak *et al.*, «Autonomous vehicles in 5G and beyond: A survey», *Vehicular Communications*, vol. 39, p. 100551, 2023, doi: https://doi.org/10.1016/j.vehcom.2022.100551.

[5] International Telecommunication Union, «IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond», *International Telecommunication Union*, vol. 2083, sep. 2015, Accedido: 7 de febrero de 2023. [En línea]. Disponible en: https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.2083-0-201509-I!!PDF-E.pdf

[6] F. Rozo-García, «Revisión de las tecnologías presentes en la industria 4.0», *Revista UIS Ingenierías*, vol. 19, n.º 2, pp. 177-192, abr. 2020, doi: 10.18273/revuin.v19n2-2020019.

[7] B. Mondal, «Artificial intelligence: state of the art», en *Recent Trends and Advances in Artificial Intelligence and Internet of Things*, vol. 172, Springer, 2020, pp. 389-425. doi: 10.1007/978-3-030-32644-9_32.

[8] E. Ceballos Velo, «Inteligencia artificial y aprendizaje automático en la gestión logística en la industria», Universidad de Cantabria, 2022. Accedido: 1 de febrero de 2023. [En línea]. Disponible en: https://hdl.handle.net/10902/26687

[9] A. Martínez Devia, «La inteligencia artificial, el big data y la era digital: ¿una amenaza para los datos personales?», *Revista La Propiedad Inmaterial*, n.º 27, pp. 5-23, jun. 2019, doi: 10.18601/16571959.N27.01.

[10] A. González Vilanova, «Métodos de machine learning en estudios biomédicos», Universidad Politécnica de Valencia, Valencia, 2019. Accedido: 1 de febrero de 2023. [En línea]. Disponible en: https://riunet.upv.es/handle/10251/127574

[11] I. H. Sarker, «Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions», *SN Comput Sci*, vol. 2, n.º 6, pp. 1-20, nov. 2021, doi: 10.1007/S42979-021-00815-1/FIGURES/13.

[12] S. Sah, «Machine Learning: A Review of Learning Types», jul. 2020, doi: 10.20944/PREPRINTS202007.0230.V1.

[13] S. Siadati, «What is unsupervised Learning», ago. 2018. doi: 10.13140/RG.2.2.33325.10720.

[14] Q. Liu y Y. Wu, «Supervised Learning», *Encyclopedia of the Sciences of Learning*, pp. 3243-3245, ene. 2012, doi: 10.1007/978-1-4419-1428-6_451.

[15] J. Si, A. G. Barto, W. B. Powell, y D. Wunsch, «Reinforcement Learning and Its Relationship to Supervised Learning», en *Handbook of Learning and Approximate Dynamic Programming*, IEEE, 2004, pp. 45-63. doi: 10.1109/9780470544785.ch2.

[16] R. S. Sutton y A. G. Barto, «The Reinforcement Learning Problem», en *Reinforcement Learning: An Introduction* , Second., A Bradford Book, 2014, pp. 1-25. Accedido: 27 de diciembre de 2022. [En línea]. Disponible en: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

[17] M. Naeem, S. T. H. Rizvi, y A. Coronato, «A Gentle Introduction to Reinforcement Learning and its Application in Different Fields», *IEEE Access*, vol. 8, pp. 209320-209344, 2020, doi: 10.1109/ACCESS.2020.3038605.

[18] K. Arulkumaran, M. P. Deisenroth, M. Brundage, y A. A. Bharath, «Deep reinforcement learning: A brief survey», *IEEE Signal Process Mag*, vol. 34, n.º 6, pp. 26-38, nov. 2017, doi: 10.1109/MSP.2017.2743240.

[19] B. Jang, M. Kim, G. Harerimana, y J. W. Kim, «Q-Learning Algorithms: A Comprehensive Classification and Applications», *IEEE Access*, vol. 7, pp. 133653-133667, 2019, doi: 10.1109/ACCESS.2019.2941229.

[20] J. Rischke, P. Sossalla, H. Salah, F. H. P. Fitzek, y M. Reisslein, «QR-SDN: Towards reinforcement learning states, actions, and rewards for direct flow routing in software-defined networks», *IEEE Access*, vol. 8, pp. 174773-174791, 2020, doi: 10.1109/ACCESS.2020.3025432.

[21] V. Mnih *et al.*, «Human-level control through deep reinforcement learning», *Nature 2015 518:7540*, vol. 518, n.º 7540, pp. 529-533, feb. 2015, doi: 10.1038/nature14236.

[22] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, y P. Li, «A Double Deep Q-Learning Model for Energy-Efficient Edge Scheduling», *IEEE Trans Serv Comput*, vol. 12, n.º 5, pp. 739-749, sep. 2019, doi: 10.1109/TSC.2018.2867482.

[23] L. Gándara Vega, «Desarrollo de un algoritmo para la evasión de obstáculos dinámicos con un robot móvil con técnica de aprendizaje con refuerzo.», Universidad Militar Nueva Granada, Bogotá, Colombia, 2020. Accedido: 6 de febrero de 2023. [En línea]. Disponible en: http://hdl.handle.net/10654/37992

[24] I. Challenger-Pérez, Y. Díaz-Ricardo, y R. A. Becerra-García, «El lenguaje de programación Python», *Ciencias Holguín*, vol. XX, n.º 2, pp. 1-13, 2014, [En línea]. Disponible en: https://www.redalyc.org/articulo.oa?id=181531232001

[25] L. Ogoti, «Why Python is Good for Machine Learning», *Section*, 4 de mayo de 2021. https://www.section.io/engineering-education/why-python-is-good-for-machine-learning/ (accedido 1 de febrero de 2023).

[26] Á. Buedo Risueño, «Desarrollo de un agente mediante Deep QLearning en un entorno de juegos de plataformas», Universidad Abierta de Cataluña, Barcelona, 2020. Accedido: 1 de febrero de 2023. [En línea]. Disponible en: https://openaccess.uoc.edu/bitstream/10609/119086/6/abuedorTFM0620memoria.pdf

[27] B. Pang, E. Nijkamp, y Y. N. Wu, «Deep Learning With TensorFlow: A Review», *Journal of Educational and Behavioral Statistics*, vol. 45, n.º 2, pp. 227-248, abr. 2020, doi: 10.3102/1076998619872761.

[28] Keras, «Keras: the Python deep learning API», *Keras*, 7 de mayo de 2020. https://keras.io/ (accedido 6 de febrero de 2023).

[29] H. Oladepo, «Nodes», *ROS Wiki*, 4 de diciembre de 2018. http://wiki.ros.org/Nodes (accedido 31 de enero de 2023).

[30] T. Foote, «Topics », *ROS Wiki*, 20 de febrero de 2019. http://wiki.ros.org/Topics (accedido 31 de enero de 2023).

[31] D. Kurzaj, «Messages», *ROS Wiki*, 26 de agosto de 2016. http://wiki.ros.org/Messages (accedido 31 de enero de 2023).

[32] T. E. Oliphant, *Guide to NumPy*, 1.ª ed. Trelgol Publishing USA, 2006. Accedido: 1 de febrero de 2023. [En línea]. Disponible en: https://web.mit.edu/dvp/Public/numpybook.pdf

[33] NumPy, «NumPy», *NumPy*, 24 de mayo de 2020. https://numpy.org/ (accedido 1 de febrero de 2023).

[34] Message Passing Interface Forum, «MPI: A Message-Passing Interface Standard Version 3.1», Knoxville, Tennessee, jun. 2015. Accedido: 1 de febrero de 2023. [En línea]. Disponible en: https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

[35] L. Dalcin y Y. L. L. Fang, «Mpi4py: Status Update after 12 Years of Development», *Comput Sci Eng*, vol. 23, n.º 4, pp. 47-54, jul. 2021, doi: 10.1109/MCSE.2021.3083216.

[36] ROS Wiki, «rospy», *ROS Wiki*, 8 de noviembre de 2017. http://wiki.ros.org/rospy (accedido 6 de febrero de 2023).

[37] TurtleBot, «What is a TurtleBot?», *TurtleBot*, 26 de marzo de 2013. https://www.turtlebot.com/ (accedido 17 de enero de 2023).

[38] TurtleBot, «What is a TurtleBot3?», *TurtleBot3*, 18 de junio de 2022. https://www.turtlebot.com/turtlebot3/ (accedido 17 de enero de 2023).

[39] N. F. Kamaruzaman y N. M. Yatim, «Integration of Mapping and Neural Network for Mobile Robot with Laser Distance Range Sensor (LDS)», *INOTEK 2021*, vol. 1, pp. 217-218, sep. 2021, doi: 10.1109/irds.2002.1041445.

[40] ROBOTS e-Manual, «TurtleBot3. Features», *ROBOTS e-Manual*, 11 de mayo de 2017. https://emanual.robotis.com/docs/en/platform/turtlebot3/features/ (accedido 6 de febrero de 2023).

[41] EDS Robotics, «▷ Cámaras Industriales: 8 tipos, características para la industria», *EDS Robotics*, 8 de octubre de 2020. https://www.edsrobotics.com/blog/camaras-industriales/ (accedido 17 de enero de 2023).

[42] EDS Robotics, «Los 12 tipos de sensores más usados: características y funciones», *EDS Robotics*, 21 de febrero de 2022. https://www.edsrobotics.com/blog/tipos-sensores-mas-usados/ (accedido 17 de enero de 2023).

[43] J. E. Salamanca Céspedes y J. N. Pérez Castillo, «LIDAR, una tecnología de última generación, para planeación y desarrollo urbano», *Ingeniería*, vol. 13, n.º 1, pp. 67-76, 2008, [En línea]. Disponible en: https://www.redalyc.org/articulo.oa?id=498850166010

[44] Intel RealSense, «Depth Camera D435», *Intel RealSense*, 25 de marzo de 2019. https://www.intelrealsense.com/depth-camera-d435/ (accedido 23 de enero de 2023).

[45] V. Tadic, A. Odry, I. Kecskes, E. Burkus, Z. Kiraly, y P. Odry, «Application of Intel RealSense Cameras for Depth Image Generation in Robotics», *WSEAS Transactions on Computers*, vol. 18, pp. 107-112, 2019, Accedido: 23 de enero de 2023. [En línea]. Disponible en: https://www.researchgate.net/publication/336495781_Application_of_Intel_RealSense_Cameras_for_Depth_Image_Generation_in_Robotics

[46] W. Woodall, «Rviz», *ROS Wiki*, 16 de mayo de 2018. http://wiki.ros.org/rviz (accedido 7 de febrero de 2023).

[47] P. Bovbel, «pointcloud_to_laserscan», *ROS Wiki*, 3 de agosto de 2015. http://wiki.ros.org/pointcloud_to_laserscan (accedido 23 de enero de 2023).

[48] M. A. Gómez Contreras, «Sincronización entre módulos para la captura de información en un robot modular tipo serpiente», Pontificia Universidad Javeriana, Bogotá, Colombia, 2017. Accedido: 25 de enero de 2023. [En línea]. Disponible en: https://repository.javeriana.edu.co/handle/10554/38700

[49] M. Pecka, «message_filters», *ROS Wiki*, 14 de agosto de 2018. http://wiki.ros.org/message_filters (accedido 25 de enero de 2023).

[50] Naciones Unidas, «Objetivos y metas de desarrollo sostenible - Desarrollo Sostenible», *Naciones Unidas* , 24 de mayo de 2022. https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/ (accedido 28 de febrero de 2023).

Anexo I. Datos numéricos

Resultados Gráfica 1

Prueba	Láser	1 cámara 90º	1 cámara 120º	1 cámara 150º	1 cámara 180º	2 cámaras
1	100,00%	71,43%	54,55%	90,00%	100,00%	88,89%
2	100,00%	60,00%	40,00%	33,33%	66,67%	84,62%
3	71,43%	90,00%	81,82%	84,62%	57,14%	66,67%
4	84,62%	76,92%	66,67%	66,67%	100,00%	71,43%
5	90,91%	57,14%	71,43%	90,00%	60,00%	87,50%
6	100,00%	90,00%	88,89%	63,64%	100,00%	75,00%
7	87,50%	66,67%	80,00%	83,33%	60,00%	88,89%
8	84,62%	23,08%	50,00%	50,00%	71,43%	66,67%
9	100,00%	53,33%	75,00%	66,67%	77,78%	90,91%
10	85,71%	72,73%	88,89%	90,91%	77,78%	88,89%
Media	**90,48%**	**66,13%**	**69,72%**	**71,92%**	**77,08%**	**80,95%**

Resultados Gráfica 2

Pruebas	Láser	1 cámara 90º	1 cámara 120º	1 cámara 150º	1 cámara 180º	2 cámaras
1	86,79%	75,00%	70,37%	83,33%	90,74%	86,79%
2	94,82%	82,54%	79,03%	68,18%	85,42%	88,67%
3	79,31%	61,20%	64,06%	82,00%	75,75%	71,01%
4	82,85%	68,97%	63,63%	73,13%	67,56%	89,28%
5	85,85%	36,65%	87,50%	74,24%	68,57%	90,47%
Media	**85,92%**	**64,87%**	**72,92%**	**76,18%**	**77,61%**	**85,24%**

Resultados Gráfica 3

Prueba	Láser	1 cámara 120º	1 cámara 150º	1 cámara 180º	2 cámaras
1	84,62%	66,67%	50,00%	66,67%	66,67%
2	81,82%	78,57%	66,67%	33,33%	75,00%
3	62,50%	80,00%	75,00%	87,50%	100,00%
4	66,67%	71,43%	76,92%	88,89%	50,00%
5	83,33%	71,43%	69,23%	72,73%	61,54%
6	84,62%	81,82%	54,55%	90,91%	90,91%
7	72,73%	60,00%	50,00%	87,50%	63,64%
8	83,33%	60,00%	70,00%	80,00%	81,82%
9	82,35%	77,78%	87,50%	100,00%	71,43%
10	91,67%	42,86%	100,00%	22,22%	83,33%
Media	**79,36%**	**69,05%**	**69,99%**	**72,97%**	**74,43%**

Resultados Gráfica 4

Pruebas	Láser	1 cámara 120º	1 cámara 150º	1 cámara 180º	2 cámaras
1	78,46%	51,02%	74,19%	84,90%	69,23%
2	85,71%	63,07%	62,68%	69,38%	86,84%
3	80,00%	70,51%	72,22%	67,21%	74,13%
4	79,69%	65,07%	69,35%	61,22%	79,03%
5	79,41%	72,97%	74,50%	70,69%	79,41%
Media	**80,65%**	**64,53%**	**70,59%**	**70,68%**	**77,73%**

Resultados Gráfica 5

Prueba	Láser	1 cámara 120º	1 cámara 150º	1 cámara 180º	2 cámaras
1	75,00%	75,00%	81,82%	88,89%	62,50%
2	70,00%	76,92%	43,75%	81,82%	66,67%
3	75,00%	75,00%	62,50%	70,00%	81,82%
4	77,78%	87,50%	63,64%	63,64%	80,00%
5	87,50%	28,57%	83,33%	90,00%	87,50%
6	83,33%	91,67%	91,67%	77,78%	90,91%
7	85,71%	69,23%	100,00%	90,00%	100,00%
8	75,00%	40,00%	81,82%	88,89%	77,78%
9	80,00%	81,82%	77,78%	77,78%	72,73%
10	90,91%	93,75%	63,64%	55,56%	75,00%
Media	**80,02%**	**71,95%**	**74,99%**	**78,43%**	**79,49%**

Resultados Gráfica 6

Pruebas	Láser	1 cámara 120º	1 cámara 150º	1 cámara 180º	2 cámaras
1	82,60%	65,27%	72,05%	74,19%	83,90%
2	67,39%	67,24%	71,66%	66,66%	59,00%
3	83,33%	57,98%	70,00%	80,90%	71,05%
4	71,83%	74,28%	68,18%	73,21%	70,90%
5	67,24%	64,51%	63,63%	59,42%	75,38%
Media	**74,48%**	**65,86%**	**69,10%**	**70,88%**	**72,05%**

Anexo II. Objetivos de Desarrollo Sostenible

En esta sección se va a explicar cuál de los 17 objetivos de desarrollo sostenible (ODS) presentan más relación con este trabajo de fin de grado. Los ODS nacieron en septiembre del año 2015 bajo la premisa de establecer unos objetivos globales a cumplir por los gobiernos, empresas privadas y sociedad en su conjunto. El fin último de este proyecto no es otra que conseguir cuidar el futuro del planeta garantizando su sostenibilidad y garantizar una vida digna de toda la sociedad, erradicando problemas como la pobreza, el hambre o la desigualdad [50].

En la imagen se pueden apreciar los distintos ODS que establecieron los líderes mundiales con el fin de alcanzar en un plazo de 15 años. De todos ellos, hay uno que está muy ligado con este trabajo desarrollado, y es el número 9 que tiene el nombre de "Industria, Innovación e Infraestructura".

Figura 33 Los 17 Objetivos de Desarrollo Sostenible.

Fuente: [50].

Este ODS está ligado a conseguir una industrialización inclusiva y sostenible a través de la aplicación de las nuevas tecnologías que se desarrollan continuamente. La relación que presenta el trabajo con este ODS está directamente relacionada con la innovación que supone el trabajo realizado. Como se ha comentado en la introducción, la aplicación de inteligencias artificiales en las empresas y en la sociedad son una realidad hoy en día, debido a los grandes y continuos avances que se están realizando en este campo.

La navegación autónoma es uno de los principales aspectos que se están desarrollando en relación con la IA, ya que son una parte fundamental para el desarrollo de las conocidas como ciudades inteligentes. Estas se entienden como la aplicación de la innovación y de la tecnología en el entorno urbano, de forma sostenible, para facilitar la gestión de la ciudad y mejorar los servicios disponibles en ella. Este trabajo, como se ha comentado, busca profundizar en la navegación autónoma con robots en el área del tratamiento de los datos, por lo que, con este trabajo se está impulsando la innovación tecnológica en este campo tan importante hoy en día.