TecnoCampus
Escola Superior Politècnica

Centre adscrit a la
upf. Universitat Pompeu Fabra Barcelona

Grado en Ingeniería Informática de Gestión i Sistemas de Información

MODELO DE APRENDIZAJE POR REFUERZO IMPLEMENTADO EN LA NAVEGACIÓN DE ROBOTS AUTÓNOMOS

Memoria Final

David Alexander Niño Chaves

Tutor: Xavier Font Aragonés

2023 / 2024

TecnoCampus
Mataró-Maresme

Dedicatoria

Este trabajo está dedicado a las personas más cercanas que se han esforzado para que yo pueda estar aquí.

Agradecimientos

Agradezco a toda la docencia del centro universitario y a los compañeros de trabajo que me han brindado la experiencia y conocimiento para realizar este proyecto.

Abstract

The main objective of this project is the implementation of a reinforcement learning (RL) model in the navigation of autonomous robots. The theoretical part reviews the definitions and functions of the various types of machine learning (ML), specializing in RL. The practice focuses on the development of the RL model with the Pytorch tool, focused on ML projects. Finally, validations are performed on real machines, extracting performance indicators that report the correct operation of the model.

Resum

Aquest projecte té com a objectiu principal la implementació d'un model d'aprenentatge per reforç (RL) en la navegació de robots autònoms. Com a part teòrica es repassen les definicions i funcions dels diversos tipus d'aprenentatge automàtic (ML), especialitzant-se en RL. La pràctica se centra en el desenvolupament del model RL amb l'eina Pytorch, enfocada en projectes de ML. On finalment es realitzen validacions en màquines reals extraient indicadors de rendiment que informen del correcte funcionament del model.

Resumen

Este proyecto tiene como objetivo principal la implementación de un modelo de aprendizaje por refuerzo (RL) en la navegación de robots autónomos. Como parte teórica se repasan las definiciones y funciones de los diversos tipos de aprendizaje automático (ML), especializándose en RL. La práctica se enfoca en el desarrollo del modelo RL con la herramienta *Pytorch*, enfocada en proyectos de ML. Donde finalmente se realizan validaciones en máquinas reales extrayendo indicadores de rendimiento que informan del correcto funcionamiento del modelo.

Índice

Índice de figuras V

Índice de tablas IX

Glosario de términos XI

1. Objetivo del proyecto 1

2. Estudio preliminar 3

2.1. Contexto 3

2.1.1. Vehículo de guiado automático, AGV 4

2.1.2. Sistema AGV Actual 4

2.1.3. Presentación de la Problemática 7

2.1.4. Tecnologías usadas en el AGV 9

2.2. Conceptos previos 11

2.2.1. Inteligencia artificial 11

2.2.2. Aprendizaje Automático 13

2.2.3. Aprendizaje no supervisado 14

2.2.4. Aprendizaje supervisado 14

2.2.5. Deep Learning, DL 15

2.2.6. Aprendizaje por refuerzo, RL 19

2.2.7. Justificación del uso de RL 26

2.3. Antecedentes 28

2.3.1. Frameworks investigados 29

2.3.2. Elección de framework 31

2.4. Implementaciones de aprendizaje por refuerzo 31

2.4.1. Proyectos básicos de ejemplo 31

2.4.2. NAF, espacio de acción continua 33

2.4.3. Deep Reinforcement Learning Hands-On 34
3. Objetivos y alcance 37
3.1. Objetivos del cliente 37
3.2. Objetivos del producto 37
3.3. Usuario final 38
3.4. Indicadores de logro de objetivos 38
3.4.1. Precisión del modelo 38
3.4.2. Tiempo de convergencia 39
3.4.3. Tiempo de predicción 39
3.4.4. Uso de los recursos del sistema 39
4. Metodología 41
4.1. Puesta a punto y configuración 41
4.2. Investigación y desarrollo 42
4.3. Análisis y conclusiones 42
4.4. Desarrollo de la interfaz visual 42
4.5. Pruebas en entornos reales 43
5. Definición de los requerimientos 45
5.1. Requerimientos funcionales 45
5.2. Requerimientos tecnológicos 45
6. Desarrollo 47
6.1. Puesta a punto del robot 47
6.1.1. Actualización componentes software 47
6.1.2. Preparación del área de pruebas 49
6.1.3. Instalación Python y dependencias 51
6.2. Desarrollo de los KPI 53
6.2.1. Precisión del modelo 53

6.2.2. Tiempo de convergencia 55
6.2.3. Tiempo entre predicciones 56
6.2.4. Uso de los recursos 56
6.3. Desarrollo del programa 56
6.3.1. Protocolo de comunicación 57
6.3.2. Arquitectura utilizada 58
6.3.3. Intercambio de datos 59
6.3.4. Modelo RL 62
6.3.5. Gestión de las predicciones del modelo 67
6.4. Pruebas realizadas 68
6.4.1. Pruebas con mouse 68
6.4.2. Pruebas con apilador 69
7. Conclusiones 77
8. Mejoras y ampliaciones 79
8.1. Mejoras en el desarrollo 79
8.2. Mejoras en las redes neuronales 79
8.3. Mejoras en el proceso de aprendizaje 80
8.4. Mejoras en el hardware 80
9. Bibliografía 82

Índice de figuras

Fig 2.1 AGV Mouse RT1500, utilizados en la fábrica de automoción Seat Martorell. Fuente propia. 3

Fig 2.2 Mapa usado por el AGV para localizarse en el entorno ya conocido. Fuente propia. 5

Fig 2.3 Esquema de sistema de navegación. Fuente: [3], modificación propia. 6

Fig 2.4 Diagrama del sistema de lazo cerrado con el controlador PID. Fuente: [4]..... 8

Fig 2.5 Diagrama de componentes UML del sistema AGV del proyecto. Fuente propia. 9

Fig 2.6 (a) Representación de un Perceptrón, (b) red neuronal de dos capas ocultas, cuatro entradas y tres salidas. Fuente: [17]..... 16

Fig 2.7 Arquitectura del modelo VGG16, usado para la detección de objetos. Fuente: [19] 18

Fig 2.8 Diagrama de la arquitectura de un sistema basado en modelo de RL. Fuente: [22]21

Fig 2.9 Tipos de aprendizaje automático, sus características y base matemática. Fuente propia, imágenes de formulación [30], [31], (1) y [25] respectivamente. 27

Fig 2.10 Visualización del entorno del problema de *Cart Pole*. Fuente: [40]..... 32

Fig 2.11 Esquema del método actor-critico, implementado en DRL. Fuente: [21]. 35

Fig 6.1 Conexión SSH desde la aplicación. Fuente propia. 48

Fig 6.2 Comandos usados para la instalación del paquete .NET 8. Fuente propia. 48

Fig 6.3 Herramienta de visualización mientras se mapea la zona de pruebas. Fuente propia. 49

Fig 6.4 Mapeo realizado para las pruebas. Fuente propia. 50

Fig 6.5 Layout desde la herramienta de dibujo ArtisterilCAD. Fuente propia. 51

Fig 6.6 Verificación de Python 3.11.0 instalado. Fuente propia. 52

Fig 6.7 Entorno virtual creado correctamente. Fuente propia. 53

Fig 6.8 Diagrama de clases simple donde se muestra la implementación de los datos de la navegación. Fuente propia. 54

Fig 6.9 Gráfica del valor de pérdida, proyecto propio de clasificación de género en imágenes. Fuente propia. 55

Fig 6.10 Diagrama del flujo de la comunicación, entre el programa de AGV y el programa del modelo RL. Fuente propia. 57

Fig 6.11 Esquema de la arquitectura hexagonal 58

Fig 6.12 Esquema de la arquitectura hexagonal aplicado en el software. Fuente propia. ... 59

Fig 6.13 Diagrama del objeto ConfigurationArgsModel. Fuente propia. 60

Fig 6.14 Diagrama del objeto StartupArgsModel. Fuente propia. 61

Fig 6.15 Diagrama del objeto StateArgsModel. Fuente propia. 61

Fig 6.16 Diagrama del objeto ModelAction. Fuente propia. 62

Fig 6.17 Diagrama del objeto FinalizeArgsModel, juntamente con el *Enum* de las razones. Fuente propia. 62

Fig 6.18 Representación visual de los puntos del trayecto que debe trazar el AGV. Fuente propia. 63

Fig 6.19 Diagrama de las propiedades de la red neuronal. Fuente propia con la herramienta *Netron* [50]. 66

Fig 6.20 Diagrama de los datos de entrada a la red neuronal. Fuente propia. 66

Fig 6.21 Gráfica del valor de perdida *Q*. Recogida por el AGV *mouse*. Fuente propia. 68

Fig 6.22 Gráfica del valor de recompensa adquirida por el entorno. Recogida por el AGV mouse. Fuente propia. 68

Fig 6.23 AGV de tipo apilador. Usado para las pruebas con el modelo RL. Fuente propia. 70

Fig 6.24 Gráfica de la pérdida del valor Q. Recogida por el AGV apilador. Fuente propia. 71

Fig 6.25 Gráfica del valor de recompensa adquirida por el entorno. Recogida por el AGV apilador. Fuente propia. 71

Fig 6.26 Gráfica de desviaciones laterales representados sobre un plano, juntamente con medidas estadísticas. Fuente propia. 72

Fig 6.27 Gráfica de puntos de parada representados sobre un plano. Fuente propia. 73

Fig 6.28 Gráfica donde muestra las medias recogidas durante la prueba, junto con las métricas estadísticas. Fuente Propia. 74

Fig 6.29 Gráfica del uso de la CPU. Fuente propia. 74

Fig 6.30 Gráfica del uso de la memoria en MB. Fuente propia. 75

Índice de tablas

Tabla 2.1 Resultados de los mejores resultados del algoritmo NAF en comparación con el agente crítico. Fuente [42]. 33

Tabla 6.1 Medidas de las acciones utilizadas por el AGV en la prueba. Fuente propia. 72

Glosario de términos

RL	Reinforcement Learning
AGV	Automatic Guided Vehicle
P_F	Posición Física
P_V	Posición Virtual
PID	Proportional-Integral-Derivative
ROS	Robot Operating System
UDP	User Datagram Protocol
PLC	Programmable Logic Controller
SCL	Structured Control Language
AI	Artificial Intelligence
AGI	Artificial General Intelligence
ML	Machine Learning
DL	Deep Learning
NN	Neural Networks
CNN	Convolutional Neural Networks
RNN	Recurrent Neural Networks
RL	Reinforcement Learning
MDP	Markov Decision Processes
DP	Dynamic Programming

TD	Temporal-Difference
DQN	Deep Q-Learning
API	Application Programming Interface
DRL	Deep Reinforcement Learning
GPU	Graphic Processing Unit
NAF	Normalized Advantage Functions
A2C	Advantage Actor-Critic method
D4PG	Distributed Distributional Deep Deterministic Policy Gradients
KPI	Key Performance Indicator
HMI	Human Machine Interface
BSD	Berkeley Software Distribution
USB	Universal Serial Bus
SSH	Secure Shell
IP	Internet Protocol
Layout	Archivo en el que se definen las trayectorias del AGV
PNG	Portable Network Graphics
JSON	JavaScript Object Notation
Link	En el contexto del *layout* son los enlaces de estaciones.
CSV	Comma-Separated Values
MB	Megabytes, unidad estándar de capacidad de memoria
LSTM	Long Short Term Memory

1. Objetivo del proyecto

Este proyecto tiene como objetivo desarrollar un modelo de aprendizaje por refuerzo (RL) en un robot autónomo móvil para gestionar la navegación, enfocado en la mejora continua en entornos industriales dinámicos.

Como marco teórico, se investiga la inteligencia artificial y sus subramas, con especial énfasis en los algoritmos, herramientas e implementaciones de RL. Dado que esta es un área relativamente nueva y en constante evolución tecnológica, se analiza estos avances para identificar las soluciones más adecuadas para el proyecto.

En la parte práctica, este proyecto busca implementar y desarrollar los algoritmos que mejor se adapten a la solución propuesta, además de medir su rendimiento mediante indicadores de eficiencia y efectividad. Estos indicadores incluyen la precisión de las predicciones del modelo, el tiempo de predicción, el tiempo de aprendizaje y el uso de los recursos del sistema. Finalmente, se garantiza la calidad y eficacia de la implementación en un entorno real para evaluar las capacidades del modelo.

2. Estudio preliminar

2.1. Contexto

Este proyecto nace de una necesidad técnica en el navegación de vehículos de guiado automático (AGV). Vehículos como se muestra en la figura “Fig 2.1” fabricados por la empresa Artisteril, especializada en implementar soluciones logísticas automatizadas a partir de vehículos robotizados en sectores industriales diversos. Entornos como la automoción, industrias farmacéuticas, plantas alimentarias, naves de almacenaje y un largo etcétera.

Fig 2.1 AGV Mouse RT1500, utilizados en la fábrica de automoción Seat Martorell. Fuente propia.

El trabajo se enmarca en la integración de una inteligencia a partir de un modelado de aprendizaje por refuerzo. Previo a la explicación de los conceptos de inteligencia artificial, aprendizaje automático y modelaje de datos. Se explicará que es un AGV y cuál es su funcionamiento básico.

El trabajo se enfoca en el desarrollo de un modelo de aprendizaje por refuerzo, para integrarlo conjuntamente al programa base del AGV. Previo a la explicación de los conceptos de inteligencia artificial, aprendizaje automático y modelaje de datos. Se explicará que es un AGV y cuál es su funcionamiento básico.

2.1.1. Vehículo de guiado automático, AGV

Como sus siglas en inglés indican, un AGV se define como un vehículo programable que tiene uno o varios sensores integrados para el guiado de forma autónoma o automática a través de un plano. Esta definición se queda pobre cuando sólo se define como el subsistema de localización del amplio sistema que engloba a un AGV. El cual contiene, además de un sistema de navegación, diversos sistemas de control como sistemas de comunicación, sistema de batería, sistema de carga y de más sistemas que lo caracteriza como solución con amplia escalabilidad.

El documento técnico [1] recoge en buen detalle tanto lo que es un AGV, como sus diferentes aplicaciones, tipos de AGV, tipo de sistemas de navegación y diferentes tipos de localización usados. Aunque el documento sea en un contexto del sector robótico de China, es una buena representación del sector en el que se encuentra la empresa Artisteril, actualmente mayormente europeo.

2.1.2. Sistema AGV Actual

En esta sección se proporcionará una breve descripción del sistema AGV diseñado y construido por Artisteril, con el propósito de establecer una base de conocimiento para la presentación del problema que abordará el proyecto.

2.1.2.1. Sistema de localización

Los sistemas de localización utilizados comúnmente en la actualidad son principalmente dos: el sistema de láser guiado por reflectores y el sistema basado en láser SLAM (Localización y Mapeo Simultáneo).

El sistema SLAM [2] logra su localización a través del continuo mapeo del entorno, utilizando diversos parámetros, entre ellos, los valores proporcionados por sensores láser, la representación del contorno mapeado y la odometría[1] del robot. El proceso de localización implica la recopilación de datos en tiempo real mediante sensores láser, que capturan información sobre el entorno circundante. Como se puede observar en la “Fig 2.2” los puntos rojos es el contorno que el sensor delantero envía, de la misma manera los puntos amarillos es la información que tiene el sensor trasero. Estos datos se utilizan tanto para construir el

[1] Sistema de localización para estimar la posición relativa con respecto a la posición inicial del AGV. Usa la información de la distancia recorrida por las ruedas y su ángulo.

mapa por primera vez, como para localizarse con respecto al mapa ya conocido mientras navega por el entorno.

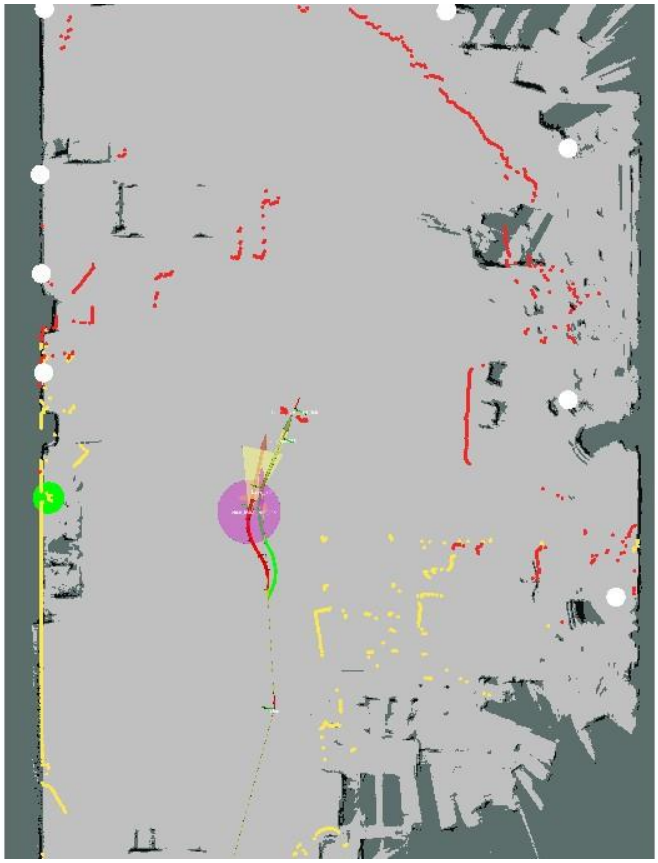

Fig 2.2 Mapa usado por el AGV para localizarse en el entorno ya conocido. Fuente propia.

A diferencia de algunos sistemas más antiguos, el sistema basado en laser SLAM ha demostrado ser sólido y confiable en entornos industriales. En el contexto de la empresa Artisteril este sistema es ideal para robots de tipo mouse de perfil bajo. Estos robots están diseñados con la finalidad de arrastrar material mediante un carro o transportarlo sobre una mesa o rodillos móviles.

El otro sistema de localización relevante se define como láser guiado por reflectores. A pesar de tener varios años, sigue siendo uno de los enfoques más sólidos, prácticos y precisos en el sector de los robots móviles. Este sistema se basa en el uso de reflectores circulares distribuidos en el plano de la instalación.

El funcionamiento de este sistema implica un sensor láser que rota 360º y lee los reflectores distribuidos en el entorno. A través de un mapa que especifica las coordenadas de los reflectores previamente definido y cargado en memoria al sensor, este realiza una triangulación de la posición en función de la información proporcionada. Este proceso devuelve de forma precisa las coordenadas del plano, consiguiendo una localización a tiempo real del robot en el espacio.

Los sistemas de localización son la base de extraer la información del entorno, de esta manera alimentar al modelo de aprendizaje que se quiere implementar. Es importante que el sistema de localización sea lo suficientemente independiente del propio AGV y que de datos fidedignos del entorno real.

2.1.2.2. Sistema de navegación

De manera simplificada, el AGV opera a partir de un sistema de localización que proporciona información sobre su posición, definida por coordenadas X, Y y un ángulo en un plano, representado como posición P_F. Utilizando esta posición y un circuito virtual predefinido, el sistema de navegación genera un nuevo punto virtual, P_V. El sistema transforma un vector basado en P_F y P_V, convirtiéndolo posteriormente en comandos para los motores, estableciendo consignas para dirección y tracción.

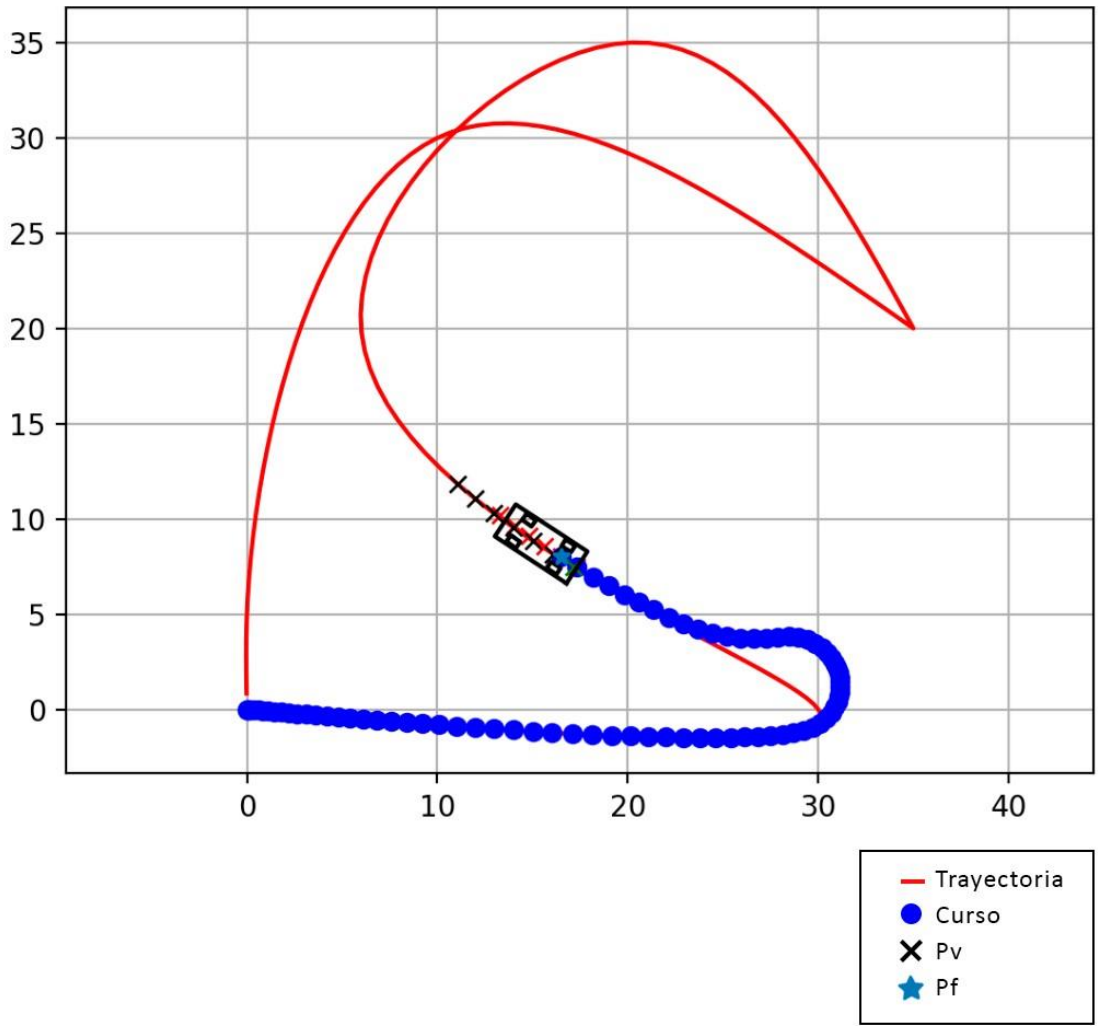

Fig 2.3 Esquema de sistema de navegación. Fuente: [3], modificación propia.

Este proceso se repite continuamente, como se puede ver en la figura esquemática “Fig 2.3”, de manera que el AGV navega con cierta precisión a lo largo del plano definido por el circuito virtual. La interacción entre el sistema de localización y el de navegación garantiza una navegación efectiva y controlada.

2.1.3. Presentación de la Problemática

La técnica que implica asignar a la máquina un punto virtual en cada instante del tiempo, con un ángulo teórico de dirección, se traduce en la posibilidad de trazar un circuito con total precisión en un entorno ideal. En este escenario, asignar al motor de dirección el valor designado para el ángulo teórico en cada punto del tiempo y espacio permitiría un seguimiento preciso de la trayectoria.

No obstante, en la realidad, esta solución enfrenta desafíos y limitaciones que afectan su precisión.

Las limitaciones más frecuentes en las instalaciones incluyen, por ejemplo, la inercia inherente de la máquina, especialmente cuando transporta carga, lo que puede resultar en posibles imprecisiones debido al aumento de peso del conjunto. Otro problema común es la presencia de imperfecciones en el suelo, que pueden ocasionar una falta de tracción en la rueda motriz; asimismo, los agujeros en el suelo pueden provocar rebotes en la máquina, afectando tanto su ubicación como su navegación óptima.

Además, estos sistemas están sujetos al desgaste de los materiales. Tanto eléctricamente, los motores pueden experimentar una pérdida de rendimiento con el tiempo, como mecánicamente, las ruedas motrices y de soporte sufren desgaste, lo que conlleva cambios significativos durante la navegación. Para abordar estas limitaciones, se recurre a técnicas de corrección de errores, siendo una de ellas el sistema de lazo cerrado.

Un sistema de lazo cerrado se caracteriza por la retroalimentación de la salida, que influye en los elementos de control. A modo de ejemplo, en el caso de la dirección de la máquina, se emite una consigna a los variadores. La salida, que incluye la información proveniente del sensor de dirección *encoder*[2] que registra la posición física de la rueda, se retroalimenta al sistema.

Adicionalmente, se implementa un controlador proporcional, integral y derivativo (PID) como elemento de control. Este controlador es un mecanismo que, a partir de la diferencia entre la retroalimentación y la consigna deseada, calcula el error. El PID utiliza tres acciones:

[2] Dispositivo que convierte el movimiento en señales eléctricas, usado para la medición de velocidad o posición.

proporcional, integral y derivativa, para extraer una nueva consigna que ajusta de manera óptima el estado del sistema.

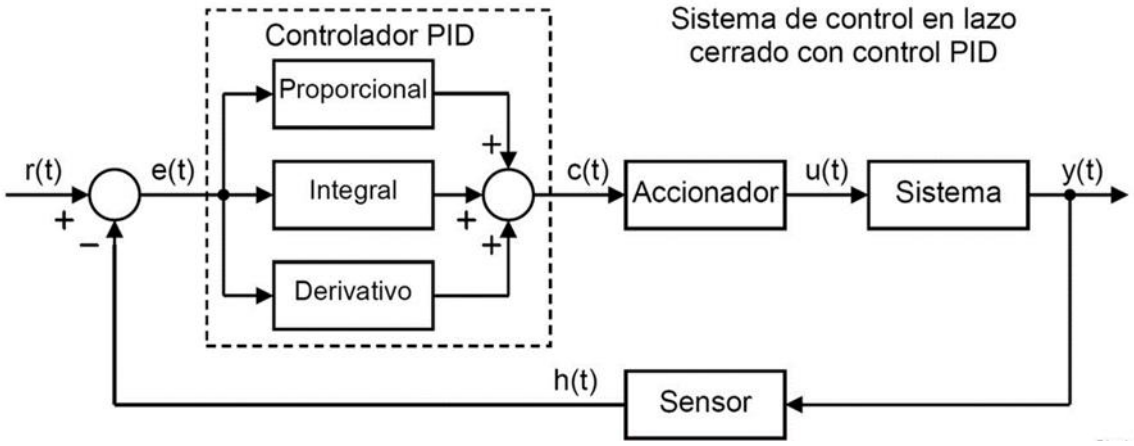

Fig 2.4 Diagrama del sistema de lazo cerrado con el controlador PID. Fuente: [4]

La “Fig 2.4” es un esquema muy representativo de cómo se integran los sistemas de lazo cerrado en conjunción a un controlador PID, usados en variedad de ocasiones para el control y corrección de error en el AGV.

En términos prácticos, el controlador PID corrige el sistema en tiempo real. La acción proporcional responde al error actual, la acción integral considera la acumulación histórica de errores y la acción derivativa anticipa las tendencias futuras. La combinación de estas acciones garantiza un control preciso y eficiente del sistema, permitiendo la adaptación continua a las variaciones en la retroalimentación y la consigna.

Cada uno de estos parámetros, tienen un inconveniente, es que se deben configurar de manera diferente para cada tipo de máquinas, según como se requiera su comportamiento. Es un punto negativo en la practicidad del método al momento de implementarlo en cada uno de los proyectos y diferentes tipos de máquinas. Además, este sistema requiere de ciertos conocimiento si se quiere conseguir la configuración óptima, lo que en la práctica en la mayoría de las ocasiones se acaba recurriendo al método de prueba y error.

Los sistemas de corrección de errores actuales desempeñan eficientemente su función en la mayoría de los casos, aunque presentan una limitación inherente en su metodología al centrarse en la reducción de errores existentes. Esta aproximación, que corrige un error que se supone no debería ocurrir, plantea una problemática fundamental.

La esencia de la problemática radica en la perspectiva negativa de corregir errores. El enfoque propuesto por este proyecto busca abordar esta cuestión mediante la utilización de la experiencia previa. En lugar de corregir un error cuando se presenta, la solución debe anticiparse al problema.

La metodología propuesta implica aprender de la experiencia previa al pasar por una zona donde se ha identificado un error. En el siguiente paso por esta área se utiliza la consigna correcta, adquirida previamente para prevenir la recurrencia del error. Este enfoque no solo busca reducir la ocurrencia de errores, sino también optimizar el rendimiento del sistema al utilizar el conocimiento previo para evitar problemas ya conocidos.

Al adoptar este enfoque basado en la experiencia, el proyecto puede contribuir en la mejora de la eficacia y la eficiencia del sistema, generando soluciones preventivas en lugar de correctivas. Este cambio de paradigma podría contribuir significativamente a la mejora continua del rendimiento del sistema en situaciones operativas específicas. Esta definición de posible solución del problema tiene una aproximación muy similar al llamado aprendizaje por refuerzo.

2.1.4. Tecnologías usadas en el AGV

Un AGV es una máquina compleja que tiene diferentes sistemas como se ha hablado anteriormente en el punto “2.1.1. Vehículo de guiado automático, AGV”. En este punto se explicarán las tecnologías usadas en los sistemas que este proyecto abordará.

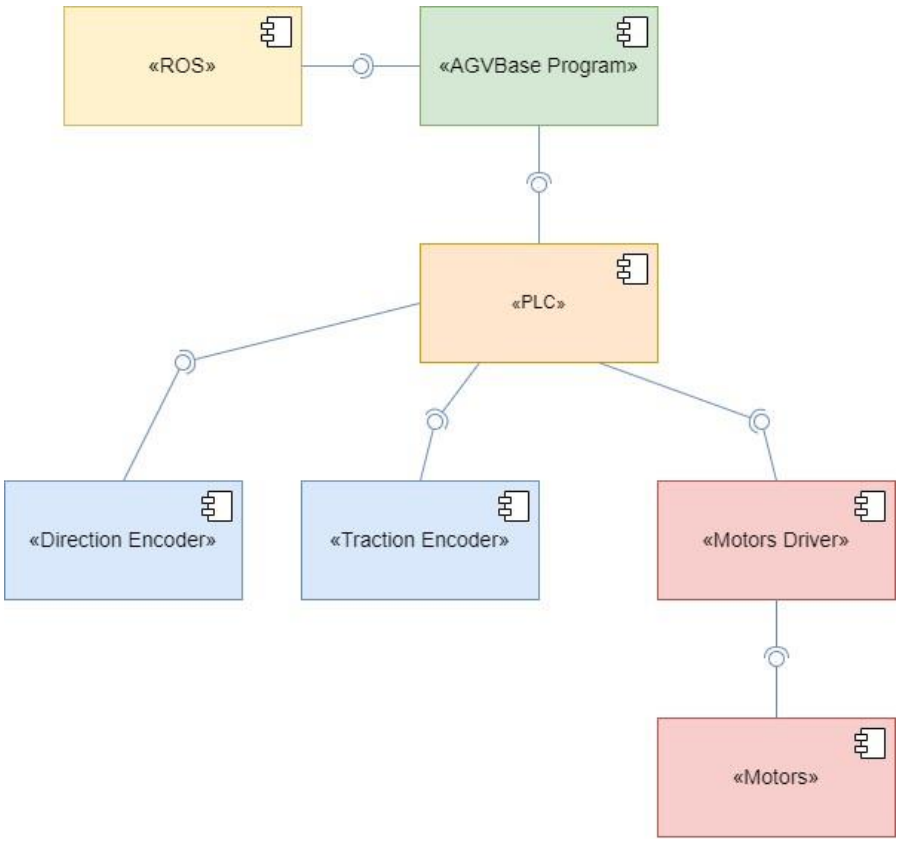

Fig 2.5 Diagrama de componentes UML del sistema AGV del proyecto. Fuente propia.

La enumeración de los elementos relevantes del AGV se enfocará a partir del diagrama de componentes de la "Fig 2.5".

El elemento central del robot es el programa base (referenciado como "*AGVBase Program*"), es el programa de control general de gran parte de los elementos. Desde los visuales como la pantalla integrada del AGV, hasta la gestión del sistema de navegación. Este programa está escrito, en su mayoría, en el lenguaje de programación *C#*, usado por la rapidez y eficiencia al ser un lenguaje compilado; además por su soporte y robustez al ser un lenguaje creado por la empresa *Microsoft*. El programa está bajo la plataforma de código abierto *NET Core*, en la actualidad con la versión 8.

Un elemento que está al mismo nivel que el programa base, se encuentra ROS (*Robotic Operative System*), definido como un conjunto de librerías, herramientas y aplicaciones para soluciones robóticas. Este "subsistema operativo" se usa como sistema de localización, proporcionando el sistema SLAM anteriormente mencionado en el punto "2.1.2.1. Sistema de localización". De manera superficial, el AGV está definido virtualmente tal y como es en las cotas físicas; ROS pide información como la odometría, los escáneres laser y su supuesta ubicación en el plano conocido; hace la computación de estas entradas y como resultado devuelve una aproximación de la localización física. Se establece una relación entre el programa base y ROS, ya que están en constante comunicación, el programa base tiene gran parte de la información que necesita ROS y el programa requiere constantemente de la localización de la máquina. Como protocolo de comunicación entre los dos componentes mencionados, utiliza UDP, ya que requiere de una latencia muy baja.

Cabe destacar que estos dos elementos, el programa base y ROS, están alojados físicamente en un ordenador industrial, equipado con un procesador *Intel i7* de décima generación de ocho núcleos. Con un sistema operativo *Linux* de distribución *Ubuntu*, ya que el subsistema ROS solo se soporta en este sistema operativo.

El siguiente componente, por orden de enumeración en el diagrama, es el PLC (*Programmable Logic Controller*). Es el elemento central a nivel de seguridad de la máquina, es el encargado de recibir las distintas entradas de la máquina, como son los *encoders* tanto de dirección como de tracción (referenciados en el diagrama). Como también accionar diferentes salidas, como por ejemplo contactores de potencia o pilotos de señalización. En muy buena parte de los proyectos realizados se instalan PLCs de la reconocida marca "*Siemens*". Dicho componente se programa a través de una aplicación de programación "*TIA Portal*", haciendo uso de un lenguaje de programación de bajo nivel llamado SCL (*Structural Control Language*). El protocolo de comunicación entre programa base y PLC es llamado S7, a través de una librería llamada "*Sharp7*".

El elemento más relevante que el PLC gestiona, con respecto a este proyecto, es el variador de los motores, referenciados en la figura como "*Motor Drivers*". Variador o variador de frecuencia, es un elemento de control, se encuentra entre la alimentación y la salida de energía hacia los motores. El variador usado es de la marca "*RoboteQ*" una marca referente

en lo que respecta a variadores, sensores y otros dispositivos de control en soluciones de AGV.

Es el encargado de controlar el movimiento de los motores conectados, en el caso de la máquina como el de la "**Error! Reference source not found.**" tiene dos motores que le sirve tanto para la tracción como la dirección de la máquina. En este caso concreto el variador tiene una comunicación vía UDP con el PLC, donde se intercambian una serie de información, entre ella las consignas de velocidad de los motores.

En resumen, el AGV se rige por un sistema jerárquico, donde cada elemento tiene una serie de responsabilidades, se gestiona o es gestionado por otros componentes. Es importante remarcar que este diagrama de componentes está enfocado en el robot que se usará en este proyecto y en diferentes máquinas pueden variar ligeramente los elementos mencionados e incluso la distribución de estos, como es el ejemplo del sistema de localización, puede ser tanto SLAM en este caso, como láser guiado por reflectores, comentados más atrás.

2.2. Conceptos previos

En esta sección, se explorarán y abordarán los conceptos fundamentales necesarios para respaldar y justificar la elección de la solución propuesta en este proyecto. El aprendizaje por refuerzo.

2.2.1. Inteligencia artificial

A lo largo de los años, el concepto de inteligencia artificial (AI) ha experimentado diversas definiciones y reinterpretaciones. En sus primeras etapas, cuando el concepto aún estaba en desarrollo, se destacó la definición presentada por el padre de la ciencia de la computación, Alan Turing.

En un seminario de 1950 [5], donde propone la pregunta fundamental "*Can machines think?*", planteó el concepto de la inteligencia artificial como un sistema capaz de pensar de manera similar al de un ser humano.

A partir de esta simple pregunta, Turing, ofrece una prueba, conocida como la "Prueba de Turing". Según esta prueba, un sistema podría considerarse inteligente si sus respuestas fueran indistinguibles de las de un ser humano durante una conversación. Esta prueba cambia el paradigma del concepto, en vez de un sistema pensante por sí mismo (hecho que le otorgaría la "inteligencia"), sería una actuación muy fidedigna a la del comportamiento humano.

Posteriormente en el artículo [6], el influyente informático John McCarthy presentó una definición de inteligencia artificial que marcó un antes y un después sobre la perspectiva inicial de Alan Turing. McCarthy propuso una definición más orientada hacia la creación y aplicación de la inteligencia artificial como una disciplina de ingeniería y ciencia.

A diferencia de la conceptualización de Turing, que se centraba en un sistema capaz de reflejar el pensamiento humano, McCarthy destacó la inteligencia artificial como un campo de estudio que involucra la creación de sistemas inteligentes, en concreto, los programas informáticos, sin caer en la interpretación de la inteligencia humana.

Actualmente, la mayor referencia en el entendimiento de la inteligencia artificial se atribuye al libro de los informáticos Stuart Russell y Peter Norvig [7]. Donde proponen cuatro potenciales definiciones, el sistema que actúa como humano, el sistema que piensa como humano, el sistema que actúa racionalmente y el sistema que piensa racionalmente. Este proyecto acoge como referencia la definición de sistema que actúa racionalmente, especificando mejor sistema como agente.

"AI has focused on the study and construction of agents that do the right thing."

"Hacer lo correcto" lo dictamina el programador de esta inteligencia, donde no solo se realiza lo correcto, sino de la mejor manera, el menor coste, maximizando los resultados y el control completo del resultado.

Acotado el concepto de AI, se definen dos tipos de inteligencia artificial, que se proceden a explicar en los siguientes capítulos.

2.2.1.1. Sistemas expertos

Los sistemas expertos, una subrama de la inteligencia artificial, también llamada inteligencia artificial débil, se define como sistema que está limitado a un campo de conocimiento experto en una o más tareas particulares. Estos sistemas carecen de la característica de escalabilidad, es decir, no están diseñados para resolver instancias de problemas fuera de su ámbito definido y son incapaces de abordar escenarios cuya resolución no está previamente establecida.

Su función es acotada, pero no por ello menos útil. Una de las ventajas más relevantes es la capacidad de reducir el tiempo en ciertos trabajos o tareas, generalmente mecánicas o repetitivas. Además, de la capacidad de análisis, muy superior e incluso incapaz a la del ser humano, en un amplio conjunto de datos. Aunque no se les puede atribuir plenamente la inteligencia, los sistemas expertos tienen la capacidad de simular la inteligencia de manera limitada y específica.

2.2.1.2. Inteligencia artificial general

Inteligencia artificial general (AGI), o también llamada inteligencia artificial fuerte, se define como una meta ambiciosa dentro del campo de la inteligencia artificial. Esta aspira al desarrollo de máquinas con capacidades que igualen o incluso superen la inteligencia humana [8]; en aspectos como conciencia, aprendizaje, resolución de problemas y planificación al futuro [9].

Es importante destacar que, hasta el momento, la AGI sigue siendo una definición principalmente teórica. Aunque sirve para tener un objetivo claro, hacia donde se quiere llegar y conseguir en este vasto campo. Lograr una máquina con una capacidad similar al humano implica superar obstáculos técnicos, éticos y filosóficos.

2.2.2. Aprendizaje Automático

El aprendizaje automático (ML), una rama clave de la inteligencia artificial, se destaca por su enfoque en el procesamiento y análisis de extensos conjuntos de datos para desarrollar la capacidad de aprendizaje de un sistema [10]. A medida que transcurre el tiempo y se captan más datos, el agente se vuelve más preciso en la ejecución de tareas específicas.

El aprendizaje automático resulta particularmente útil en problemas donde no se dispone de un algoritmo definido para obtener una solución. Un ejemplo es la discriminación de correos electrónicos como “Spam” o a los que no lo son [11]. Como entrada se encuentra el correo electrónico, un conjunto de caracteres, en un formato específico e incluso imágenes. En este caso, la entrada consiste en el correo electrónico en sí, un conjunto de caracteres o incluso imágenes, y la salida deseada es una clasificación binaria: verdadero o falso, es o no es un correo basura.

Dado que no existe un algoritmo definido que pueda transformar de manera directa un correo electrónico en una respuesta que distinga de “Spam” o no, el aprendizaje automático se convierte en una herramienta ideal. Utilizando el conocimiento de miles de correos identificados como basura y miles que no lo son, puede extraer automáticamente un algoritmo a través del procesamiento y aprendizaje de datos.

El potencial del aprendizaje automático radica en aprender de la experiencia dispuesta, adaptarse a nuevos datos, a su propio entorno, realizar futuras predicciones, abordar problemas complejos y dinámicos, ofreciendo soluciones a través de su propia inferencia y con la capacidad de proporcionar una solución óptima.

2.2.3. Aprendizaje no supervisado

El aprendizaje no supervisado es una rama del aprendizaje automático que se centra en manejar entradas no etiquetadas o no clasificadas, que en esencia no es más que puro ruido no estructurado [12]. Su objetivo principal es descubrir patrones o grupos inherentes en los datos sin el conocimiento previo de las etiquetas o clasificaciones.

El modelaje del aprendizaje no supervisado es usado para 3 tareas principales [13]. Agrupamiento en clústers o "*Clustering*" es una de las funciones del aprendizaje no supervisado, trata de agrupar los datos recibidos en grupos basándose en sus similitudes, diferencias o cualquier parámetro definido. Los algoritmos de "*Clustering*" pueden ser clasificados en unos pocos tipos; exclusivos, superpuestos, jerárquicos y probabilísticos.

Otro uso del aprendizaje no supervisado es la regla de asociación. Esta se basa en la detección de relaciones entre variables en el conjunto de datos recibido. Es especialmente útil en sistemas de recomendación, por ejemplo, productos que puede comprar por cestas de compras anteriores, o mostrar películas que se asemejan a películas ya vistas.

Como tercer y último uso se encuentra la reducción de dimensionalidad. Esta es una técnica usada para la reducción de los datos recibidos, tanto cantidad como dimensionalidad de estos, con la capacidad de mantener la integridad de los propios datos y así conseguir una mayor manejabilidad del conjunto de datos, por ejemplo, para su visualización.

El aprendizaje automático no supervisado es una herramienta eficaz y muy útil en la exploración y comprensión de datos no etiquetados, ofreciendo soluciones y aplicaciones prácticas en diversos campos donde la complejidad y diversidad de los datos requieren enfoques flexibles y adaptables.

2.2.4. Aprendizaje supervisado

El aprendizaje supervisado, también conocido como aprendizaje automático supervisado, se basa en el entreno de la máquina a partir de un conjunto de datos que se le proporciona de forma clasificada [14]. Estos datos están organizados en formatos de entrada o entradas y salidas deseadas. La máquina se entrena a partir del conjunto de datos proveído. Cuando se presenta un valor de entrada, el modelo aprendido se ejecuta y devuelve una predicción de salida que se acerca lo máximo posible a los datos proporcionados.

Este tipo de aprendizaje se distingue por la necesidad de supervisar adecuadamente los datos de entrenamiento, ya sea de forma automática o manual. La precisión de los resultados del modelo será representativa de la calidad de los datos de entrenamiento. Estos datos pueden cambiar o modificarse a medida que el modelo obtiene experiencia de respuestas. Si las salidas del modelo no cumplen con la precisión deseada, se pueden corregir y asignarlas en

los datos de entrenamiento para un posterior reajuste de su función. Como si de un sistema de lazo cerrado se tratara.

El tipo de problemas más comunes de este aprendizaje son dos, clasificación y regresión [15]. Los problemas de clasificación ayudan a categorizar los datos de entrada según una clasificación definida en el conjunto de datos de entrenamiento. En el apartado de explicación del "Aprendizaje automático" doy como ejemplo el problema de detección de correo basura, si es o no lo es.

Otro ejemplo más relacionado con el proyecto, y que resulta más cercano a un nivel de experiencia práctica, es el de clasificar si la imagen captada por el AGV representa los agujeros de un palé o no. A través de un extenso conjunto de imágenes de entrenamiento, previamente clasificadas como agujeros de palé y no agujeros, el AGV se posiciona frente a un palé, captura una fotografía con su cámara incorporada y transmite esta imagen al modelo. El modelo devuelve la clasificación y, además, indica la ubicación precisa de los agujeros mediante ocho puntos en el plano de la imagen, correspondientes a los rectángulos formados por estos agujeros.

En cuanto a la regresión, se trata de un tipo de análisis estadístico que define la correlación entre una variable, denominada dependiente, y una o más variables independientes. La principal utilidad de este análisis es la capacidad predictiva de la variable dependiente basada en las variables independientes. La regresión es útil cuando se busca analizar cuáles son las variables más relevantes en cierta solución de problemas y, por ende, también se utiliza para descartar parámetros menos útiles. Un ejemplo de regresión podría ser la predicción de la edad, como variable dependiente, en función del nivel de colesterol en la sangre, considerado como variable independiente [14]. Concluido el análisis, se determina una correlación del 95% entre estas dos variables, aunque se reconoce que las predicciones pueden mejorarse.

2.2.5. Deep Learning, DL

El Deep Learning (DL) también conocido por su traducción al español como aprendizaje profundo, es un tipo de aprendizaje que emplea redes neuronales artificiales [16]. Estas redes neuronales buscan simular la funcionalidad de aprendizaje del cerebro humano. Al igual que las neuronas en el cerebro humano se conectan entre sí, los perceptrones [17] o neuronas artificiales también se conectan, formando un entramado organizado por capas.

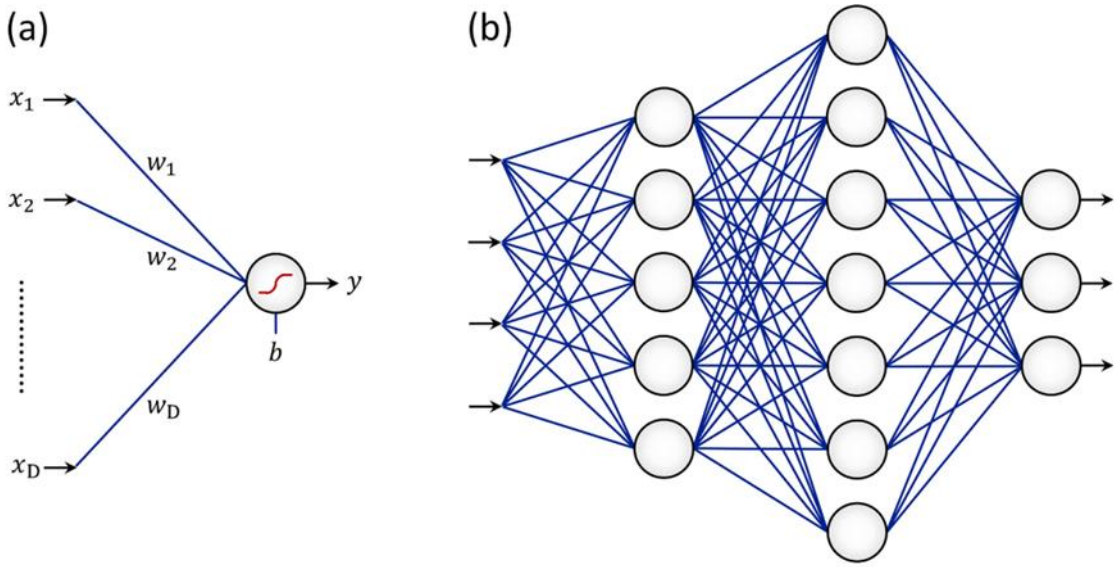

Fig 2.6 (a) Representación de un Perceptrón, (b) red neuronal de dos capas ocultas, cuatro entradas y tres salidas. Fuente: [17]

Los perceptrones constituyen la unidad mínima en la red neuronal artificial, y se representan como se muestra en la "Fig 2.6" (a). La formulación del perceptrón se expresa como (1):

$$y = f\left(\sum_{i=1}^{D} w_i\, x_i \; + b\right) \tag{1}$$

La formulación establece que y será la salida del perceptrón y se obtendrá mediante la suma de todas las entradas. Esta suma es el resultado de multiplicar cada elemento del vector x por el peso w correspondiente de esa entrada, más un *bias*[3] b. Dependiendo de los valores de las variables descritas se determinará si la neurona se activa o no, afectando en las entradas de los siguientes perceptrones o directamente en la salida del resultado, como se observa en la "Fig 2.6" (b).

Gracias a este concepto básico de una red neuronal, es posible comprender mejor cómo funciona el conjunto. La principal ventaja de DL sobre ML radica en el tipo de conjunto de datos que puede manejar. Mientras que en ML se requiere de datos preprocesados para que aprenda ciertos tipos de entradas con las salidas deseadas, DL puede procesar un gran número de conjunto de datos sin una previa estructuración o categorización experta.

[3] Propiedad intrínseca del perceptrón, definida como la facilidad con la que el perceptrón puede activarse.

La clave del aprendizaje profundo se define en que el sistema aprende los datos mediante un procedimiento de aprendizaje de propósito general [18]. Este es un aprendizaje completamente automático. Un ejemplo claro de este procedimiento son los denominados "*captchas*" y las solicitudes típicas como "Marca la cabeza de la tortuga". La respuesta se pasa a un sistema de aprendizaje profundo, que analiza la fotografía y el área marcada donde se supone que está la cabeza. A través de repeticiones, las redes neuronales se ajustan para calibrar los pesos y *bias*, para finalmente obtener mayor precisión.

La principal ventaja del aprendizaje profundo radica en su implementación en numerosas soluciones relacionadas con la inteligencia artificial y sus diversas ramas. Sin embargo, es una metodología de alto nivel que exige un sólido conocimiento. No solo debido a su gran potencial de automejora y precisión, sino también porque requiere una potencia de cálculo que, dependiendo de la cantidad de datos que se utilice, no está al alcance de cualquier institución o persona.

Las redes neuronales (NN) utilizadas en el aprendizaje profundo se diseñan de diversas maneras y formas, conformando una representación no lineal en diferentes capas, como se observa en la "Fig 2.6" (b). La primera capa siempre representa las entradas de datos, la última se define como la salida de la red, y las capas centrales, llamadas capas ocultas, realizan la función de convertir las entradas en niveles más abstractos. Cuantas más capas ocultas haya, mejor será la predicción y la precisión del sistema de aprendizaje.

Existen diferentes tipos de redes neuronales utilizadas en el aprendizaje profundo. Uno de los más simples es el denominado "*forward propagation*". De hecho, este se ha explicado como la red donde las entradas atraviesan las neuronas que van computando, enviando la salida a la entrada de la siguiente neurona, hacia adelante, de ahí el nombre. Como tipo complementario de "*forward*" se encuentra "*backward propagation*" La intencionalidad del procedimiento de la red neuronal es realizar el camino contrario una vez el modelo ha dado su predicción. A partir del error o respuesta correcta que se proporciona (como en el aprendizaje supervisado), se dan los pasos contrarios por las neuronas activadas previamente, de manera que se vayan modificando los pesos (w) de estas, para minimizar el error la próxima vez.

En niveles más altos de complejidad, se encuentran las redes neuronales convolucionales (*ConvNets* o CNN) y las redes neuronales recurrentes (RNN). El diseño de las redes

convolucionales parte de la necesidad de procesar datos provenientes de múltiples matrices, como es el caso de las imágenes. Estas redes tienen cuatro puntos clave: conexiones locales, pesos compartidos, el uso de una capa llamada "*pooling*" y la combinación de estas, juntamente con las convolucionales. De manera superficial, estas redes descomponen los datos de entrada, de matrices bidimensionales a matrices más simples. Los datos atraviesan las capas convolucionales y de "*pooling*", deconstruyéndose en datos más abstractos y simples. Por ejemplo, una imagen donde aparezca un gato se va descomponiendo en la nariz del gato, las orejas, los ojos, etcétera. Explicado de una forma más técnica, la matriz de pixeles de la imagen del gato se va comprimiendo en matrices más pequeñas y simples. De esta manera, las CNN pueden detectar patrones o características en las imágenes que se analizan.

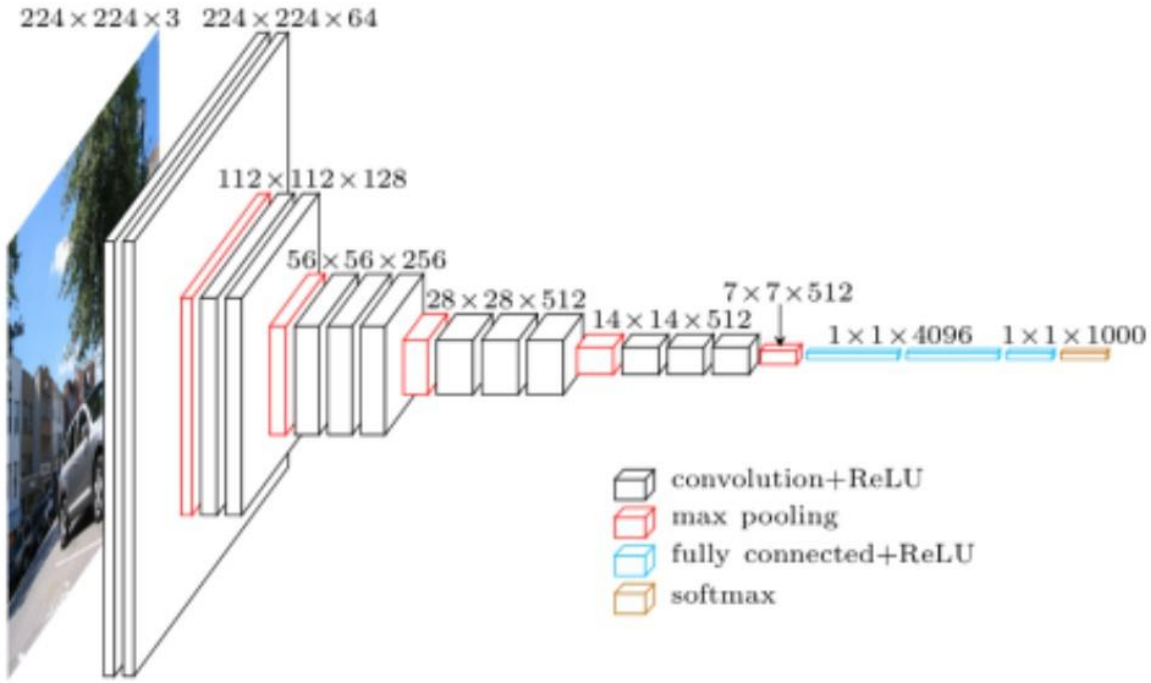

Fig 2.7 Arquitectura del modelo VGG16, usado para la detección de objetos. Fuente: [19]

En la "Fig 2.7" se puede observar un esquema de la arquitectura del modelo de aprendizaje profundo "*VGG16*" para la detección y clasificación de objetos, se puede reconocer las diferentes capas usadas tanto capas convolucionales combinándolos con las capas "*pooling*" anteriormente mencionadas.

En cuanto a las RNN, son redes neuronales utilizadas mayormente para el procesamiento del lenguaje natural y la generación de texto hablado. Este tipo de redes neuronales son usadas para tipos de datos secuenciales, lo cual estados de entradas anteriores tienen relevancia con las entradas se procesan. La clave de estas redes está en que cada neurona contiene un vector de estado, que representa las entradas que han pasado anteriormente. De manera lógica, la red mantiene un contexto, por ejemplo, una frase de entrada. Así se consigue que no solo comprenda una palabra de la frase, sino el contexto completo de la oración.

2.2.6. Aprendizaje por refuerzo, RL

Este es el último punto teórico con respecto a la explicación de los métodos de aprendizaje automático que abordará este proyecto.

Reinforcement Learning (RL), en su traducción al español aprendizaje por refuerzo, se define como la forma de entrenar a un agente para que trabaje de manera racional, óptima y eficaz dentro de un entorno [20]. Este agente descubre cuáles son las mejores acciones que debe tomar mediante una política establecida. La manera de determinar si una acción es correcta o incorrecta se rige por una recompensa. El agente cuenta con información sobre su entorno, las consecuencias de las acciones tomadas, la puntuación de recompensa asociada a dicha acción y, por ende, el estado del objetivo asignado a este agente [21].

En el aprendizaje por refuerzo, se encuentran dos elementos principales: el agente y el entorno. Sin embargo, en el sistema de aprendizaje se identifican cuatro subelementos encargados del funcionamiento global para alcanzar el objetivo final: la política, la señal de recompensa, el valor de función y el modelo del entorno.

La política es, en esencia, el comportamiento que adopta el agente en su entorno. Constituye el núcleo del “pensamiento” del agente, dictaminando las acciones que debe tomar en respuesta a la percepción del estado del entorno en el que se encuentra. Generalmente, la política del agente debe ser estocástica, lo que significa que las acciones que tome deben tener probabilidades asociadas. Esta probabilidad se entenderá más adelante en los términos de exploración y explotación.

La señal de recompensa se define como el objetivo final del problema de aprendizaje por refuerzo. El agente busca maximizar esta recompensa, que es proporcionada por el entorno. Esta recompensa es un número que será mayor si las decisiones de la política son correctas y menor si las acciones tomadas resultan en un estado desfavorable del entorno. Una vez captada esta señal, la política se modifica para aumentarla y alcanzar el objetivo. La señal de recompensa, generalmente, debe ser una función estocástica con respecto a las acciones realizadas y al estado correspondiente del entorno.

El valor de función es una versión complementaria a la señal de recompensa. Mientras que la recompensa devuelve un valor de la acción inmediata realizada, el valor de función proporciona información sobre cuán bueno es el estado actual del entorno y los posibles estados siguientes. Este valor determina cuánto es el máximo de recompensa que el agente puede aspirar en el futuro desde el estado actual.

En ocasiones, se pueden encontrar situaciones en las que una acción número uno tiene una recompensa inmediata grande, pero una acción número dos, aunque su recompensa no sea mayor que la de la acción uno, tiene un valor de función superior. En este escenario, el agente debe valorar mejor la acción dos para obtener una recompensa total mayor. Es importante destacar que este valor es, en realidad, una predicción de futuras recompensas, lo cual presenta uno de los mayores desafíos en el aprendizaje por refuerzo; lograr una función óptima para estimar correctamente este valor.

Por último, el cuarto elemento en el sistema de aprendizaje por refuerzo es el modelo del entorno. En esencia, este modelo es una simulación del entorno físico en el que trabaja la máquina. El objetivo del modelo es, mediante la información actual del entorno y la acción que el agente tomará, predecir el siguiente estado del entorno y estimar la señal de recompensa que este enviará. Es una de las maneras de prever si la acción tomada es la mejor, evitando depender completamente del error que se genera al probarla en el entorno físico.

Este cuarto elemento no es estrictamente necesario en el sistema; de hecho, en RL se realiza esta distinción. Por un lado, el sistema de modelo libre, que es una forma simple de lograr los objetivos mediante prueba y error mientras se interactúa en el entorno. Por otro lado, a

pesar de su mayor complejidad, en los últimos años se ha apostado por el segundo tipo, el sistema basado en el modelo del entorno, como presenta en el siguiente diagrama “Fig 2.8”. Este enfoque hace que el proceso sea más planificado y predictivo, contribuyendo a una mejor eficiencia en la toma de decisiones del agente.

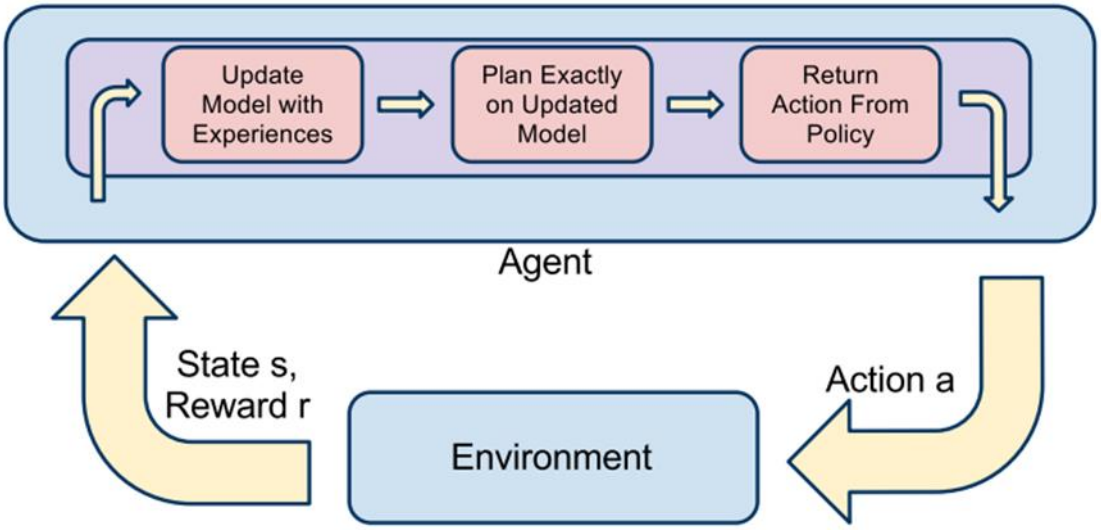

Fig 2.8 Diagrama de la arquitectura de un sistema basado en modelo de RL. Fuente: [22]

Otra de las claves fundamentales del aprendizaje por refuerzo es que considera el problema a resolver como el objetivo principal dentro de un entorno incierto. En contraste con otros enfoques de aprendizaje, donde se entrena al sistema con conjuntos de datos sin necesariamente comprender el uso de estos datos en el entorno, el RL se centra en la toma de decisiones para lograr un objetivo específico.

En el aprendizaje no supervisado, el valor del sistema radica en reconocer diferentes grupos o patrones en una diversidad de datos, pero no necesariamente implica comprender el significado intrínseco de los datos. Por otro lado, el aprendizaje supervisado realiza predicciones o estimaciones basadas en datos que ya están previamente clasificados con entradas y salidas correctas, sin especificar necesariamente cuál es el propósito o la finalidad de estos datos.

La característica distintiva de un sistema de RL es su capacidad para comprender plenamente el entorno, tomar decisiones con la finalidad de cambiar el estado de este y alcanzar un

objetivo directo. Este enfoque orientado a objetivos lo convierte en una herramienta poderosa para abordar problemas complejos en entornos dinámicos y cambiantes.

Pese a las ventajas y a los potentes métodos de resolución que ofrece el aprendizaje por refuerzo, también presenta una serie de inconvenientes. Como se mencionó en la explicación del elemento de RL, la política, se encuentran los términos de exploración y explotación. Estos dos modos son fundamentales en el comportamiento de las decisiones que toma el sistema. La explotación implica aprovechar la experiencia, cuando una acción pasada ha conseguido una buena señal de recompensa, reeligiéndola nuevamente. Por otro lado, la exploración representa un intento de la política del sistema de obtener recompensas aún mayores, llevándolo a tomar acciones que no han sido seleccionadas previamente o diferentes a las anteriores, por lo tanto, que pueden no ser las mejores a efecto inmediato, pero pueden llevar a un camino con mayores recompensas.

El desafío principal radica en el correcto equilibrio entre estos dos modos. El dilema entre exploración y explotación es crucial y debe ser gestionado cuidadosamente para que la máquina pueda alcanzar sus objetivos de manera óptima y eficaz. Encontrar la proporción adecuada entre aprovechar lo aprendido y buscar nuevas oportunidades es esencial para el éxito del sistema en entornos dinámicos y complejos.

Finalmente, para comprender técnicamente el RL, es necesario explicar en qué se fundamenta la gran mayoría de los algoritmos. Posteriormente, se enumerarán algunos de estos algoritmos y se proporcionará una breve descripción de cada uno.

2.2.6.1. Proceso de decisión de Márkov, MDP

Markov Decision Processes (MDP) es una de las formalizaciones clásicas de un proceso de control de toma de decisiones secuencial. En este enfoque, la toma de acciones no solo influye en los estados y recompensas actuales, sino también en el cómputo total de los caminos futuros. En el contexto del RL, MDP se define formalmente como el conjunto de elementos que se han explicado previamente: los estados del entorno, las acciones a tomar según la política, la secuencia de transiciones y la señal de recompensa.

En la formalización de MDP, se representa como el conjunto de elementos (S, A, T, R) [23], donde:

- S es un conjunto de estados finitos.
- A es el conjunto de acciones finitas a tomar.
- T es la función de transición de estados.
- R es la función recompensa.

Este marco proporciona una estructura formal para modelar problemas de toma de decisiones secuenciales en entornos de RL para encontrar políticas óptimas. Todos los algoritmos explicados a continuación comparten el uso de MDP como base para su implementación y funcionamiento.

2.2.6.2. Programación dinámica, DP

La programación dinámica o dynamic programming (DP), en su traducción al inglés, es un conjunto de algoritmos usados para la optimización de problemas complejos, que se subdividen en problemas más simples. Solucionando estos subproblemas recursivamente, se van guardando las soluciones óptimas. De esta manera que se consigue solucionar el problema total de la manera más eficaz y eficiente [24].

En RL, se usa DP para optimizar las políticas, centrándose en encontrar la mejor manera de tomar decisiones secuenciales a lo largo del tiempo, u optimizar el modelo del entorno, para conseguir predicciones de un gran número de transiciones de estado. Sin embargo, la clave para lograr optimizar las políticas es mejorando las predicciones de valores de función.

2.2.6.3. Ecuación de Bellman

La ecuación de Bellman o *Bellman equation*, traducido al inglés, es una de las bases de RL. Esta ecuación define la relación entre un estado y la recompensa esperada en un determinado punto en el tiempo [25]. Esta ecuación es usada como método básico para subdividir los problemas complejos en DP. En el contexto de RL, es usado para describir cual es el máximo de valor de recompensa que se puede obtener en un estado determinado.

Además, es una de las bases de cálculo de los valores *V* y los valores *Q*, en los algoritmos *Q-learning* y *DQN*, definidos más adelante. Las ecuaciones son definidas de la siguiente manera:

Ecuación de Bellman para los valores *V* se define como (2).

$$V(s) = E\pi[\, Rt + 1 + \gamma V(st + 1) \mid st = s \,] \quad (2)$$

Ecuación de Bellman para los valores Q se define como (3).

$$Q(s, a) = E[\, Rt + 1 + \gamma a' maxQ(st + 1, a') \mid st = s, at = a \,] \quad (3)$$

2.2.6.4. Método de Montecarlo

El método de Montecarlo o en su traducción al inglés, *Monte Carlo Methods*, se refiere a una clase de algoritmos computacionales que utilizan la aleatorización de números para obtener resultados deseados. Una definición más analógica, implica representar un problema mediante una población hipotética y captar un número aleatorio de muestras de esta población. A través de estas muestras aleatorias, se obtiene una estimación estadística de los parámetros relevantes del problema en cuestión [26].

Este enfoque se utiliza en una variedad de contextos, como la simulación de sistemas complejos, la integración numérica y la resolución de problemas que involucran aleatoriedad o incertidumbre. En el contexto de (RL), los métodos de Montecarlo se aplican basándose en la media generada por el algoritmo. En estos métodos, aunque pueda tener un modelo, este lo único que debe generar son las transiciones de los estados, y no predicciones de todas las posibles transiciones. A través de múltiples transiciones de estado y diversos valores de recompensa obtenidos, el algoritmo ajusta y mejora sus estimaciones de la función de valor. Así proporcionando una forma de evaluar y actualizar la calidad de las acciones tomadas en los estados diversos.

2.2.6.5. Temporal difference learning

Temporal-Difference Learning (TD) es una clase de algoritmos del tipo de sistemas de modelo libre, previamente explicados. TD combina características de los métodos de Monte Carlo y los algoritmos de programación dinámica (DP). Al igual que los Métodos de Monte Carlo, TD se aprovecha de la experiencia, ya que no requiere un modelo explícito del entorno como en DP. Sin embargo, en comparación con DP, TD realiza optimizaciones de las predicciones mediante un método llamado "*bootstrapping*".

En términos simples, el "*bootstrapping*" implica que las predicciones pueden optimizarse justo antes de que se realice la señal de salida. En el contexto de un AGV durante la navegación, por ejemplo, se podría predecir que el AGV tomará ciertos grados de dirección en la siguiente curva. Utilizando DP, un intervalo de tiempo antes de llegar a la curva, se

puede optimizar el valor de la dirección, ya que se tiene una información más clara sobre cómo se encuentra el entorno actualmente.

2.2.6.6. Q-learning

Q-learning es uno de los principales algoritmos dentro del contexto de RL. Usando las características de TD, es del tipo de sistema de modelo libre, *Q-learning* se basa en asignar valores Q en cada tupla de estado y acción en todo el entorno adquirido [27]. El valor Q representa una predicción de recompensa que el entorno devolverá. El elemento de la política no está tan presente, ya que se guía por el mayor valor Q. El algoritmo se defina de la siguiente manera (4):

$$Q(S_t, A_t) \leftarrow\ Q(S_t, A_t) +\ \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, A_t) - Q(S_t, A_t)] \quad (4)$$

La clave del algoritmo para su correcto funcionamiento es la actualización continua de cada valor a lo largo del camino elegido. En cada transición, los valores Q se actualizan, como ya se ha explicado en el punto anterior "2.2.6.3" con la ecuación de Bellman, así proporcionando una mejor predicción en la siguiente iteración. De esta forma, al principio el agente siempre tendrá unos valores parecidos y el modo de trabajo será la exploración. En cuanto vaya descubriendo y mejorando los valores, únicamente tendrá los caminos óptimos para llegar a su objetivo, usando de esta manera la explotación.

2.2.6.7. Sarsa

El algoritmo *Sarsa* es una modificación del anterior algoritmo explicado, *Q-learning*. Mientras *Q-learning* actualiza sus valores Q con la recompensa predicha más alta, *Sarsa* actualiza los siguientes valores, pero generando las acciones de la próxima transición. Se comprende mucho mejor con la definición formal (5) y comparándola con la formalización de *Q-learning*.

$$Q(S_t, A_t) \leftarrow\ Q(S_t, A_t) +\ \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (5)$$

Como se puede observar, este algoritmo realiza la elección del siguiente valor Q mediante la generación de la siguiente acción predicha mediante la política que se ha definido (γ). El nombre proviene del uso de la quíntupla usada para realizar las actualizaciones de los valores $(S_t, A_{t,} R_{t+1}, S_{t+1}, A_{t+1})$.

2.2.6.8. Deep Q-learning

El algoritmo *Deep Q-learning* (DQN), como su nombre indica, basado en el algoritmo *Q-learning* y utiliza redes neuronales para refinar las estimaciones de los valores de función. Como se explicó en la sección "*2.2.5 Deep Learning, DL*", las redes neuronales requieren entrenamiento para afinar las predicciones de los valores objetivos designados, en este caso, los valores Q del algoritmo.

En DQN, se emplea la técnica de repetición de la experiencia para facilitar el entrenamiento de la red neuronal. Esta experiencia se refiere al uso de una memoria para almacenar las trayectorias tomadas por el proceso de MDP [28]. En cada iteración, se guardan los estados, acciones, recompensas y el siguiente estado en la memoria de experiencia. De esta manera, se logra entrenar a la red neuronal para optimizar y mejorar las estimaciones en las siguientes decisiones.

2.2.6.9. Proximal policy optimitzation, PPO

Proximal Policy Optimization (PPO) se basa en mejorar la política de forma iterativa, ajustando cambios mientras toma decisiones para aumentar la precisión del sistema [29]. La clave de las modificaciones radica en que no son en grandes pasos; en cada actualización, se realizan cambios pequeños para asegurar la estabilidad del modelo.

El objetivo de PPO es maximizar la función objetivo, que es una combinación de la función de valor y la política. Generalmente, PPO hace uso de redes neuronales entrenadas de manera supervisada para mejorar iterativamente esta función objetivo mientras explora e interactúa con el entorno.

PPO es uno de los algoritmos que actualmente está dando mejores resultados. Tanto es así que la reconocida empresa *OpenAI* utiliza este algoritmo en su inteligencia artificial. Creando así modelos basados en PPO que juegan a videojuegos como *Dota 2*, venciendo a jugadores profesionales. Además, se ha aplicado con éxito en el control de diferentes automatismos robotizados.

2.2.7. Justificación del uso de RL

Después de haber analizado un gran número de definiciones, metodologías y algoritmos durante esta sección de "2.2 Conceptos previos". Se concluyen varios puntos. Este proyecto tiene como objetivo dotar de un aprendizaje a un robot móvil, del cual ha de ir aprendiendo en base a la experiencia que se le va presentando a lo largo de su vida útil, o al menos gran parte de ella. Por ello el aprendizaje automático es uno de los términos generales a los que

se ha de encaminar este trabajo. En la "Fig 2.9" se resumen muy brevemente los tipos explicados.

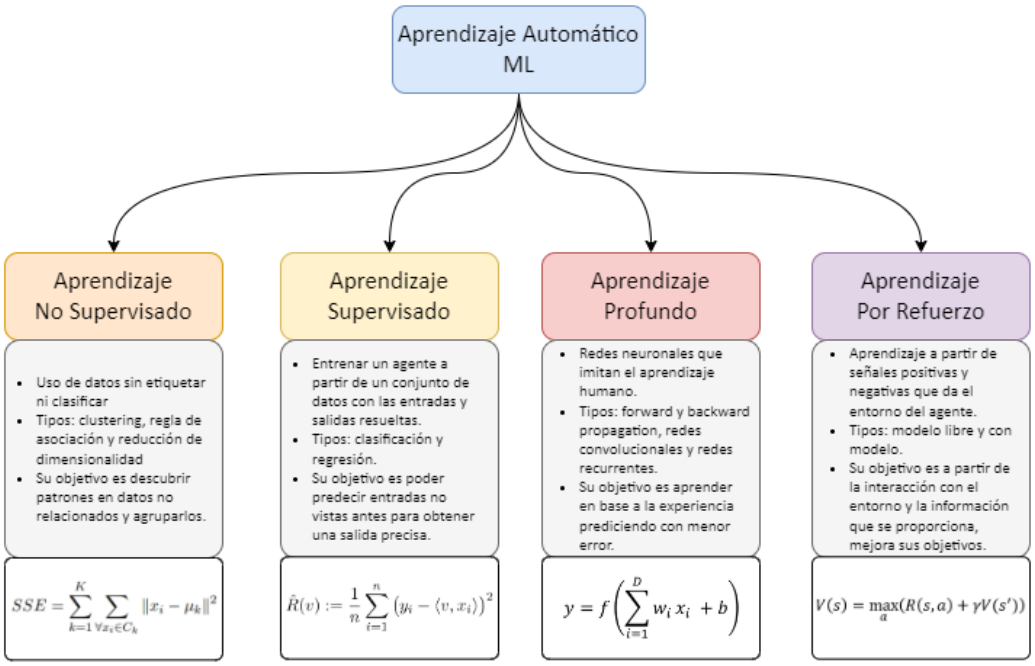

Fig 2.9 Tipos de aprendizaje automático, sus características y base matemática. Fuente propia, imágenes de formulación [30], [31], (1) y [25] respectivamente.

Una vez claras las subcategorías de ML, la primera de estas, explicada en el punto "2.2.3. Aprendizaje no supervisado", no se puede hacer uso de este aprendizaje en el contexto del proyecto. No hay ningún objetivo de agrupación o clasificación de datos variados y diferentes en los objetivos de la navegación del AGV. Por lo tanto, queda completamente descartado este planteamiento.

Como segundo subgrupo de aprendizajes automáticos, se encuentra el explicado en el punto "2.2.4. Aprendizaje supervisado". Este tipo de aprendizaje se reconoce que podría ser un enfoque interesante; ya que permite proporcionar fácilmente datos de entrada y salidas deseadas. Sin embargo, este enfoque sería bastante manual y dependiente del sistema actual. Dependiente por la necesidad en la correcta configuración de las máquinas y mantener el circuito idóneo. Se desestima por completo, ya que en los entornos reales prácticamente nunca van a mantenerse estables, al ser un campo muy dinámico y cambiante. Ya que conllevaría ir manteniendo el sistema en constante supervisión y entrenamiento por los tales cambios.

El aprendizaje supervisado sería una metodología impracticable y poco rentable a largo plazo, por lo tanto, queda descartada como practica en el uso por sí solo. A pesar de esto, se sugiere que podría ser útil como apoyo del sistema elegido.

Como siguiente aprendizaje descrito, se encuentra en el punto "2.2.5 Deep Learning, DL". Este es un método que tiene unas categorías de aprendizaje plenamente automáticas. El uso de las redes neuronales en este trabajo lleva un nivel de complejidad bastante alto. Es una metodología bastante enfocada al apoyo y complemento con otros modelos de aprendizaje. Por lo tanto, de la misma manera que el aprendizaje supervisado, el uso aislado de DL sería complejo y complicado. No por ello se descartado como soporte al uso en los algoritmos de aprendizaje por refuerzo, técnica llamada DRL (*Deep Reinforcement Learning*), haciendo uso de redes neuronales para conseguir los objetivos.

La decisión final y el enfoque principal del proyecto se centrarán en el aprendizaje por refuerzo RL. Esta elección se justifica por su definición, que se encuentra en el punto "2.2.6. Aprendizaje por refuerzo, RL". Aunque sea un punto apartado de la jerarquía del ML, es una subcategoría de esta.

En el contexto de este proyecto, el agente será el robot AGV, y su objetivo será navegar por un circuito previamente mapeado, el entorno. El uso de las diferencias entre los puntos físicos y virtuales determinará la señal de recompensa devuelta por el entorno, junto con otros aspectos como la velocidad del robot.

Dado que RL es uno de los campos más complejos dentro de ML, se requerirá una investigación más profunda para configurar adecuadamente el entorno y la máquina. Además, será necesario implementar los algoritmos presentados en secciones anteriores. En el próximo punto, se llevará a cabo un análisis de cómo se realizarán estas implementaciones.

2.3. Antecedentes

En esta sección de antecedentes se analizarán los *frameworks* que brindan soporte para la implementación de ML y más concretamente de RL en diversos sistemas.

Donde finalmente se realiza la elección y justificación del *framework* que se usará en este trabajo basándose en la investigación realizada.

2.3.1. Frameworks investigados

2.3.1.1. TensorFlow

TensorFlow, desarrollado originalmente por *Google* [32], es una de las bibliotecas más conocidas e importantes en el ámbito de ML, especialmente en DL. Es una plataforma de extremo a extremo, lo que significa que ofrece una amplia gama de herramientas que se pueden conectar y utilizar en todo el proceso de aprendizaje automático.

TensorFlow proporciona conjuntos de datos para diversos casos de uso, y se puede utilizar tanto en dispositivos físicos compatibles como en servicios en la nube para el procesamiento de datos. Además, *TensorFlow* está respaldado por la API de más alto nivel, *Keras*. *TensorFlow* opera con una estructura de datos llamada tensores. Dicha estructura de datos consiste en matrices multidimensionales optimizadas para operaciones complejas en ML. Estos tensores pueden instanciarse a partir de cualquier tipo de dato primitivo, como booleanos, enteros o números decimales. Además, *TensorFlow* hace especial uso del compartición de memoria entre dispositivos, como la CPU y la GPU del ordenador donde se esté ejecutando el tensor, permitiendo así maximizar el rendimiento del *hardware*.

En cuanto a las implementaciones específicas de aprendizaje por refuerzo, *TensorFlow* cuenta con ejemplos enfocados en el aprendizaje profundo, como el ejemplo de programación del algoritmo DQN para el entorno *Cartpole* [33]. Este ejemplo puede servir como guía para el desarrollo del algoritmo en el contexto del proyecto. Para otros algoritmos que no implican aprendizaje profundo, es posible realizar implementaciones, pero podría requerir una investigación adicional.

2.3.1.2. PyTorch

PyTorch es uno de los *frameworks* más reconocidos tanto en el ámbito académico como en la industria. Similar a *TensorFlow*, *PyTorch* es una plataforma de extremo a extremo, pero se especializa en la implementación de modelos de aprendizaje profundo en el lenguaje Python. La nueva versión de *PyTorch* (*PyTorch 2.0*) introduce una funcionalidad adicional llamada "*torch.compile*", que permite compilar modelos escritos en *Python* al lenguaje *C++*, mejorando la velocidad y la dinamicidad [34]. Como también se ha visto con *TensorFlow*, *PyTorch* también basa la implementación con tensores.

PyTorch es conocido por su fuerte integración con *Python* y ofrece compatibilidad y funcionalidades extensas con tarjetas gráficas, tanto de *NVIDIA* como de *AMD*, lo que aumenta la potencia de los modelos de aprendizaje. Es uno de los *frameworks* más utilizados en artículos académicos y publicaciones de desarrollo [35], lo que lo convierte en una excelente opción para el proyecto. La amplia adopción de *PyTorch* facilita el acceso a diversos ejemplos e implementaciones que pueden servir como punto de partida para el desarrollo del proyecto.

2.3.1.3. Keras

Los desarrolladores de *Keras* han lanzado recientemente una nueva versión, *Keras 3*, que se presenta como una herramienta de aprendizaje automático compatible con múltiples *frameworks*. *Keras* permite elegir entre diferentes motores, como *JAX*, *TensorFlow* y *PyTorch*, desde un mismo código base [36]. Esta versatilidad hace que sea una herramienta potente, ya que se puede utilizar una variedad de herramientas y arquitecturas de los diferentes *frameworks* disponibles para el desarrollo del modelo.

Aunque *Keras* ofrece flexibilidad y opciones, la curva de aprendizaje puede ser complicada debido a la amplia gama de posibilidades y la alta escalabilidad que proporciona. Sin embargo, el uso de *Keras* puede ser una buena elección, especialmente si se busca la capacidad de cambiar entre diferentes *frameworks* según las necesidades del proyecto y poder realizar un estudio más amplio de optimización.

2.3.1.4. OpenAI Gym

OpenAI Gym, también conocido simplemente como *Gym*, es una biblioteca de herramientas diseñada especialmente para RL [37]. *Gym* proporciona una extensa colección de entornos y problemas que pueden implementarse en diversos *frameworks*, como *PyTorch* o *Keras*, mencionados anteriormente. Aunque *Gym* no es en sí un *framework* para la implementación de modelos de aprendizaje, ofrece una amplia variedad de ejemplos prácticos para implementar dichos modelos, como el mencionado ejemplo de *Cartpole* [33], que utiliza la herramienta *Gym* para obtener el entorno del ejemplo.

Además de las herramientas que ofrece, *Gym* es un sitio *web* donde los usuarios pueden compartir los resultados de los desarrollos implementados. Esto permite comparar experiencias, identificar posibles mejoras y revisar las optimizaciones realizadas en algoritmos similares para un mismo problema.

2.3.1.5. Pearl

Pearl ("*A Production-Ready Reinforcement Learning Agent*") es el *framework* de código abierto creado por *Meta*. Se trata de una plataforma de extremo a extremo especializada en RL, enfocada para la implementación de un agente versátil que consiga los objetivos de los problemas reales [38]. Además, adopta la filosofía el diseño completamente modular de las diversas capacidades que tiene el agente a implementar. Capacidades como la exploración inteligente, sensibilidad al riesgo, restricciones de seguridad y la experiencia conseguida con el tiempo.

Pearl se ha construido a partir de *PyTorch*, por lo cual hereda todas sus características, como la compatibilidad con GPUs de *NVIDIA* y *AMD*. Una de sus ventajas es que es un *framework* realmente reciente y que se mantiene por personas expertas en el sector, que están soportadas por una gran empresa detrás como es *META*.

2.3.2. Elección de framework

Después de llevar a cabo la investigación y análisis de las diversas opciones en el campo del aprendizaje automático, se ha determinado que uno de los *frameworks* más robustos y ampliamente utilizados es *PyTorch*. Esto se debe tanto al lenguaje base que emplea, con el cual personalmente ya se cuenta con cierta familiaridad y experiencia, eliminando así la barrera de aprendizaje desde cero de un nuevo lenguaje de programación. Lo que queda por investigar es el desarrollo de los algoritmos de aprendizaje por refuerzo mencionados en el punto “2.2.6. Aprendizaje por refuerzo, RL”.

Además, se observó que *PyTorch* es utilizado por numerosos investigadores en todo el mundo para llevar a cabo sus implementaciones de aprendizaje por refuerzo. No solo se utiliza *PyTorch* en el desarrollo de soluciones, sino que también se emplea como una capa base con un entorno de nivel superior, como en el caso de *Keras*, siendo *PyTorch* uno de los motores de implementación elegibles entre varios. También es la base de la biblioteca de herramientas de la compañía *Meta*, con su propio *framework* desarrollado, *Pearl*.

Se ha descartado tanto el uso de *Keras* como de *Pearl* al tratarse de "sobre plataformas" de la base principal que es *PyTorch*, aunque no por ello son inferiores. Además, se ha tenido en cuenta la curva de aprendizaje asociada a estas plataformas. No solo es necesario entender la plataforma utilizada, sino también comprender al menos las capas subyacentes. Por esta razón, se consideró el *framework TensorFlow*, pero finalmente se descartó debido a que *PyTorch* ofrece una mejor compatibilidad con el uso de tarjetas gráficas, lo que sugiere una mayor escalabilidad al momento de implementar modelos más complejos.

2.4. Implementaciones de aprendizaje por refuerzo

En esta sección se investiga y define las distintas implementaciones, con el fin de obtener una base sólida para comenzar el desarrollo del trabajo.

2.4.1. Proyectos básicos de ejemplo

Dado que este proyecto es un primer contacto con el extenso mundo de RL, se han buscado primeras referencias en proyecto de ejemplo. Al hacer uso de *PyTorch*, su sitio *web* oficial

es una de sus primeras fuentes consultadas, donde se pueden encontrar ejemplo de los proyectos más famosos y sencillos, como es el de *Cartpole*.

En el enlace [39], se encuentra un ejemplo desarrollado con el algoritmo *DQN*, brevemente explicado en el punto "2.2.6.8". Este ejemplo describe de manera sencilla el objetivo del modelo: mantener un péndulo invertido recto el mayor tiempo posible, donde encuentra encima de un carro, únicamente utilizando únicamente dos acciones posibles, mover el carro a la derecha o a la izquierda. Es un buen punto de partida para entender y comprender la metodología del uso de las diversas herramientas que proporciona PyTorch. Este proyecto utiliza la librería *Gymnasium*, mencionada en la sección "2.3.1.4", donde se desarrolla el entorno base del proyecto.

Investigando más proyectos de ejemplo se encuentra, en el enlace [40], la implementación del mismo problema del carro, esta vez desarrollado con el algoritmo PPO, investigado previamente en el punto "2.2.6.9".

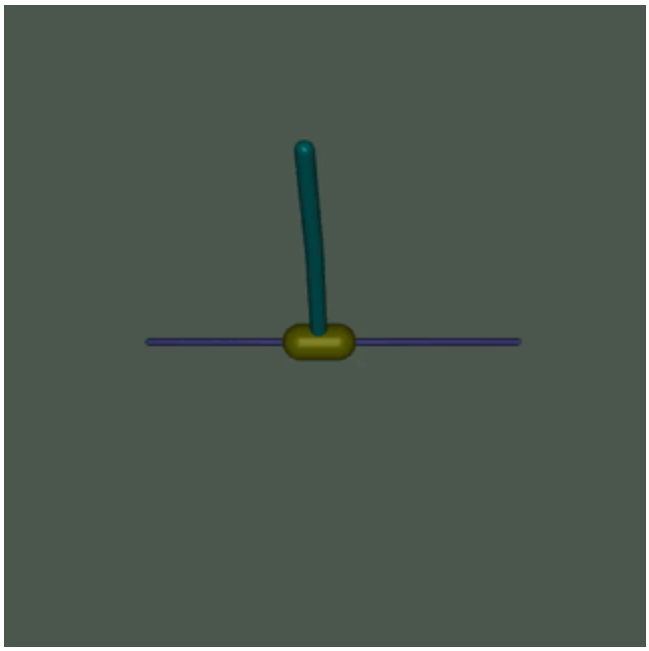

Fig 2.10 Visualización del entorno del problema de *Cart Pole*. Fuente: [40]

Dado que en este proyecto se debe crear desde cero el entorno del AGV durante la navegación en el plano, se ha encontrado la siguiente referencia a seguir [41], donde se usa el mismo problema base del péndulo, pero realizando el entorno con la herramienta "*EnvBase*" de la librería *torchrl*. Este paquete está dedicado a las implementaciones de aprendizaje por refuerzo en *PyTorch*.

Obtenidas las referencias para la creación del entorno, se enfoca en el diseño del agente y la política. El objetivo del modelo es predecir los grados de consigna del motor de dirección y así navegar por el entorno en el trayecto definido. A diferencia de los ejemplos anteriores, que no tienen la misma naturaleza de acciones a tomar, el problema del péndulo presenta dos posibles acciones. En cambio, el modelo del AGV debe poder elegir entre una infinidad

de acciones (el máximo de grados que admita la rueda). Al tratarse de un ángulo decimal, se trata de un entorno con espacio de acción continua.

2.4.2. NAF, espacio de acción continua

Se tomará como referencia el siguiente artículo [42], el cual explica dos algoritmos complementarios. El primero es una variante del algoritmo *Q-learning*, denominado NAF (*Normalized Advantage Function*). Este algoritmo tiene la ventaja de poder trabajar con espacios de acción continuos, es decir, con valores de predicción infinitos, que es la naturaleza del proyecto. La idea principal se resume en que facilita la determinación analítica del mejor valor de acción al descomponer la función *Q* en componentes más manejables.

Además del algoritmo NAF, el artículo también describe una técnica de aprendizaje seguro en el entorno robótico. El método que normalmente se usa para el aprendizaje del modelo implica mostrar tanto acciones buenas como malas. Sin embargo, en una máquina donde el hardware puede dañarse, este método podría acarrear severas consecuencias. Lo que se propone es añadir una técnica denominada "*imagination rollouts*". Ésta consiste en añadir virtualmente acciones incorrectas con ruido, logrando así obtener tanto acciones buenas como malas de manera segura. Esto resulta en un proceso de aprendizaje seguro y eficiente.

El artículo hace referencia a pruebas con distintos problemas de RL, mostrando los mejores resultados en cuanto a la convergencia del modelo en comparación con la técnica del actor-crítico, definida en el punto "2.4.3". En la "Tabla 2.1" se puede observar una clara ventaja en la reducción de episodios con NAF junto con la técnica de aprendizaje de los "*imagination rollouts*". Es decir, el número de iteraciones necesarias para que el modelo aprenda y obtenga buenas recompensas se reduce significativamente.

Domains	-	DDPG	episodes	NAF	episodes
Cartpole	-2.1	-0.601	420	-0.604	**190**
Reacher	-2.3	-0.509	1370	**-0.331**	**1260**
Peg	-11	-0.950	690	**-0.438**	**130**
Gripper	-29	1.03	2420	**1.81**	**1920**
GripperM	-90	-20.2	1350	**-12.4**	**730**
Canada2d	-12	-4.64	1040	**-4.21**	900
Cheetah	-0.3	**8.23**	**1590**	7.91	2390
Swimmer6	-325	-174	220	**-172**	**190**
Ant	-4.8	-2.54	2450	-2.58	**1350**
Walker2d	0.3	**2.96**	**850**	1.85	1530

Tabla 2.1 Resultados de los mejores resultados del algoritmo NAF en comparación con el agente crítico. Fuente [42].

Además del artículo, útil para entender la base del algoritmo y los resultados obtenidos con dichas técnicas, se ha encontrado un código abierto en la plataforma *GitHub* [43], desarrollado con el paquete de herramientas *PyTorch*. Este código se tomará como base para el desarrollo del agente y la política, ya explicada previamente. Sin embargo, deberá adaptarse, ya que utiliza el entorno de *CartPole*, visto en los anteriores proyectos de ejemplo. Además, será necesario añadir los KPI necesarios para medir los objetivos propuestos.

2.4.3. Deep Reinforcement Learning Hands-On

El libro Deep Reinforcement Learning Hands-On [44] está destinado al aprendizaje de la herramienta *PyTorch*. Este título contiene diversas implementaciones con explicaciones tanto teóricas como prácticas y referencias a un repositorio de *GitHub* [45] libre para realizar pruebas. Se enfoca en implementaciones de DRL en diversos entornos, como la automatización de páginas *web*, *chatbots* y robótica; este último muy relevante en el trabajo actual.

Además de contener explicaciones teóricas para comprender las bases de RL, el *framework* de *PyTorch* y los diferentes tipos de algoritmos aplicables a diversos problemas, también muestra los resultados prácticos del código presentado. Es una excelente base para aprender en profundidad sobre RL.

Este libro contiene un capítulo llamado "*Chapter 17: Continous Action Space*", que se enfoca en la problemática ya comentada en el punto "2.4.2" sobre la infinidad de acciones posibles que el agente RL puede tomar. Se presentan tres técnicas diferentes para abordar esta problemática: *advantage actor-critic method* (A2C), *deterministic policy gradients* y *distributed distributional deep deterministic policy gradients* (D4PG).

Todos estos métodos se basan en el método actor-crítico, que consiste en dos componentes: el actor, encargado de aprender la política que debe seguir el modelo RL, seleccionando correctamente las acciones para alcanzar los objetivos, y el crítico, encargado de evaluar la política actual generada por el actor, devolviendo un error TD (*Temporal Difference*) al actor para que este reajuste su política basado en dicho error. En la "Fig 2.11" se puede observar un esquema para entenderlo de manera más visual. Esta técnica es esencialmente utilizada en situaciones donde las acciones que el agente RL puede tomar son muy amplias o infinitas.

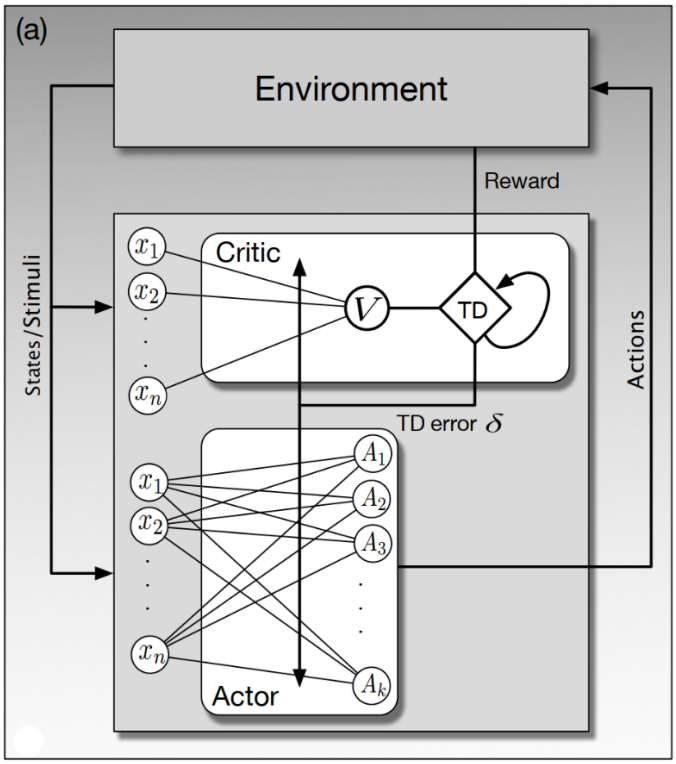

Fig 2.11 Esquema del método actor-critico, implementado en DRL. Fuente: [21].

En definitiva, el libro es una excelente base, no solo para aclarar aspectos esenciales de RL, sino también como ayuda en la implementación del agente. Al tener referencias directas con código práctico, es una buena forma de comprender cómo funciona la parte teórica y realizar pruebas locales en el entorno del AGV.

3. Objetivos y alcance

3.1. Objetivos del cliente

- Lograr precisión en el movimiento de la máquina autónoma.
- Optimizar la eficiencia de la navegación de la máquina.
- Configurar la máquina de forma correcta con una formación mínima.
- Identificar rápidamente posibles fallos relacionados con el modelo de aprendizaje.
- Habilitar la activación o desactivación del modelo de manera ágil y sencilla, sin afectar su operación normal.
- Visualizar una interfaz atractiva y con información relevante para el monitoreo del modelo.

3.2. Objetivos del producto

- Implementar un modelo de aprendizaje por refuerzo para mejorar la navegación en el AGV.
- Implementar este modelo únicamente en los robots con movimiento de triciclo.
- Integrar el programa para que pueda transmitirse de manera fluida con el sistema base de la máquina, asegurando su funcionamiento.
- Garantizar la compatibilidad del programa tanto en sistemas Windows como en Linux.
- Establecer la estandarización de parámetros configurables mediante archivos, permitiendo la modificación del comportamiento del modelo.
- Implementar el programa de manera que el modelo tome las acciones en tiempo real.
- Crear indicadores para el monitoreo y validación efectiva del modelo de aprendizaje.
- Diseñar una interfaz sencilla para la monitorización y configuración básica a nivel de operario.
- Elaborar una interfaz avanzada para extraer métricas y gráficos, facilitando el monitoreo continuo del modelo.

3.3. Usuario final

El usuario final de este producto se segmenta en dos tipos. El primero es el usuario básico, que se encuentra en las instalaciones y corresponde al cliente que ha adquirido el AGV. Este usuario recibirá una formación básica sobre el funcionamiento interno del modelo, cómo activar o desactivar el modelo, el entendimiento de las alarmas del sistema en relación con el modelo de aprendizaje y el proceso para resolver posibles problemas. Es crucial tener en cuenta este perfil de usuario, ya que, a pesar de su conocimiento básico, será la persona que esté más tiempo y en mayor contacto con el AGV y, por lo tanto, con el producto. Se debe realizar un estudio especializado en el diseño de accesibilidad, posibles flujos de trabajo tanto favorables como desfavorables e incluso considerar posibles pruebas "A/B" en cuanto a la interfaz de usuario se refiere.

En cuanto al segundo usuario, este será un perfil más avanzado. Además de la capacitación básica, recibirá una explicación superficial de cómo funciona el modelo de aprendizaje por refuerzo en el sistema del AGV. Adquirirá conocimientos sobre los parámetros configurables del modelo, la funcionalidad y las consecuencias de modificarlos en el comportamiento de la máquina. También comprenderá los indicadores, métricas y gráficas para la trazabilidad del modelo, así como el proceso de aprendizaje, la monitorización a lo largo de la implementación en el entorno y cómo se valida el correcto funcionamiento. A diferencia del usuario básico, el usuario avanzado estará presente en las primeras instancias del modelo y será responsable de ajustar y validar el modelo. Por lo tanto, es de vital importancia realizar un estudio exhaustivo de los flujos de trabajo, verificando el diseño juntamente con personas que tengan este perfil, para obtener perspectivas diversas.

3.4. Indicadores de logro de objetivos

En esta sección se explicarán diversas ideas sobre cómo medir el éxito del proyecto, mediante varios *KPIs* que indiquen diferentes aspectos en la implementación del modelo de aprendizaje de los *AGVs*.

3.4.1. Precisión del modelo

El indicador principal que se tendrá en cuenta para evaluar la mejora de la navegación será la precisión de la máquina mientras se mueve por el entorno. Para medir esta precisión, se establecerá una media de la desviación lateral de la máquina durante la navegación. Además, se medirán las desviaciones en los puntos de parada entre las coordenadas virtuales y las físicas (P_v, P_f). Este indicador será crucial para determinar uno de los objetivos principales de este proyecto.

La desviación lateral proporcionará información sobre cuán cerca o lejos la máquina se encuentra del camino planificado, lo que reflejará la precisión del modelo. La comparación entre las coordenadas P_v y físicas P_f en los puntos de parada permitirá evaluar la precisión de las decisiones y acciones del sistema en los momentos dónde se requiere mayor precisión.

3.4.2. Tiempo de convergencia

La evaluación del tiempo de convergencia será otro indicador relevante en el momento de elección como algoritmo óptimo. Se medirá el tiempo que el modelo tarda en converger y estabilizar las acciones decididas. Cuanto menor sea el tiempo de convergencia, se asumirá una mayor eficiencia del sistema frente a cambios futuros.

Un tiempo de convergencia más rápido indica que el AGV puede adaptarse rápidamente a nuevas condiciones o situaciones, lo que es esencial en entornos dinámicos. Este indicador también refleja la capacidad del modelo para aprender de manera eficiente y tomar decisiones óptimas en un tiempo razonable.

3.4.3. Tiempo de predicción

La lógica del AGV implica que su programación es a tiempo real. Para lograr este objetivo correctamente en el modelo de aprendizaje del proyecto, será crucial tener en cuenta el tiempo entre predicciones. Este tiempo se refiere a la diferencia entre el momento en que el entorno proporciona al agente o modelo las entradas junto con la recompensa, y el momento en que el agente realiza la acción devuelta para interactuar con el entorno. Para que sea lo más funcional posible, este tiempo debe ser casi nulo y se debe mejorar con el tiempo a medida que el agente trabaja, ya que las predicciones deberán ser menos frecuentes y más precisas.

3.4.4. Uso de los recursos del sistema

La eficiencia en el uso de recursos es una característica relevante de cualquier programa informático, incluido en el sistema del AGV. Cuantos menos recursos consuma del sistema operativo, más beneficioso será para el cómputo total. Sin embargo, es importante contrastar este gasto de los recursos con los indicadores anteriores y realizar una ratio que evalúe la eficacia del agente en relación con los recursos consumidos.

Aun así, se debe tener muy en cuenta ya que, si requiere de gran parte del sistema, puede acarrear ciertos problemas de seguridad del AGV. Si el programa del modelo RL consume demasiado y llega a saturar el ordenador, algunos elementos físicos de la máquina pueden

quedar en un estado que no debería en funcionamiento normal. Como por ejemplo son los controladores de los motores, si estos reciben una orden de movimiento y el ordenador se queda “congelado” el controlador seguirá con la consigna recibida. Hasta que un elemento de seguridad de más bajo nivel actúe (paro de emergencia o escáneres de seguridad) el robot no se detendría.

La ratio de eficacia del agente entre los recursos consumidos proporcionará una métrica de evaluación muy útil en el momento de analizar la capacidad del AGV. Una ratio más alto indicará que el AGV está logrando una mayor eficacia en términos de rendimiento y logro de objetivos en relación con la cantidad de recursos utilizados.

4. Metodología

Este proyecto utilizará *Agile* [46] una metodología basada en lapsos de tiempo, llamados *sprints*, donde se definen unos objetivos de trabajo y llegados al final del *sprint* se entrega a la persona interesada para la revisión.

Se definirán cuatro puntos clave para alcanzar los objetivos del proyecto:

1. Puesta a punto y configuración
2. Investigación y desarrollo
3. Análisis y conclusiones
4. Desarrollo de la interfaz visual
5. Pruebas en entornos reales

En los siguientes puntos se definen el proceso que llevará cada uno de estos.

4.1. Puesta a punto y configuración

Este será el punto de partida del proyecto. Antes del desarrollo de los algoritmos, se deberá establecer un espacio de trabajo. Este espacio debe ser un área pequeña donde el robot pueda moverse libremente y que sea lo más estable posible para las pruebas de los diferentes algoritmos, evitando así discrepancias por la propia zona de trabajo.

Una vez asegurado el lugar físico para el uso del robot, se deberá realizar el mapeo del espacio de la manera más precisa posible. Para las pruebas se decide usar el tipo de robot *mouse* “Fig 2.1”, ya que es relativamente pequeño, práctico para esta clase de pruebas. Este es un buen candidato para comenzar puesto que el objetivo es implementarlo en máquinas con tipo de movimiento triciclo.

Posteriormente, será necesario poner a punto la máquina, lo cual incluye la instalación del software necesario. En el caso de este proyecto, donde se ha elegido el *framework* de PyTorch, también será necesario instalar los correspondientes paquetes de Python. Además, se deberá crear el *layout* para la navegación del AGV.

Con estos pasos ya completados, se podrán llevar a cabo pruebas con el sistema actual. Estas pruebas permitirán obtener información sobre la eficacia actual del sistema. Los datos por extraer están definidos en el punto “3.4. Indicadores de logro de objetivos”. De esta manera, se podrán comparar los resultados con los desarrollos que se llevarán a cabo en la siguiente etapa del proyecto, “4.3. Análisis y conclusiones”.

4.2. Investigación y desarrollo

Esta etapa es una de las más importantes del proyecto, ya que implica la investigación y la implementación de los algoritmos mencionados en el punto "2.4". Además, incluye el diseño de las comunicaciones entre los dos programas: el programa base del AGV y el modelo en *Python*. Se debe analizar el diseño de software de estos programas para asegurar la escalabilidad del proyecto a lo largo del tiempo.

Durante este proceso, se hará uso principalmente de la metodología *Agile*. Se establecerán *sprints* con plazos de una semana o diez días (considerando que cada algoritmo es un poco más complejo que el anterior). Al final de cada sprint, se definirán las conclusiones basadas en los datos recopilados y se documentarán las dificultades encontradas a lo largo del proceso.

El uso de la metodología Agile dará paso a la utilización de herramientas de seguimiento de procesos como Trello o Monday.com. Estas herramientas facilitarán la gestión y la organización del proyecto, proporcionando una mejor trazabilidad del trabajo realizado. Además, permitirán visualizar de manera efectiva los puntos donde se han encontrado más problemáticas a lo largo del tiempo transcurrido.

4.3. Análisis y conclusiones

Después de realizar las pruebas con los diferentes algoritmos propuestos y recopilar datos durante las pruebas, se llevará a cabo una comparación exhaustiva para identificar la eficiencia, los fallos y posibles áreas de mejora. Para realizar esta comparación, se crearán tablas que contengan los diferentes *benchmarks*, con el objetivo de tener una visión clara de las fortalezas y debilidades de cada algoritmo. Con esta información, se podrán tomar decisiones informadas sobre cuál o cuáles son los algoritmos más adecuados para cumplir con los objetivos del proyecto.

4.4. Desarrollo de la interfaz visual

Para conseguir los objetivos propuestos y que la implementación del modelo de aprendizaje sea funcional en casos reales, es necesaria una interfaz gráfica de la cual extraer información en caso de mal funcionamiento del sistema. Los trabajos por realizar serán los siguientes:

- Desarrollar una sección con valores de configuración a modificar según los casos de uso.

- Mostrar en gráficas o valores diferentes parámetros de información relevante para el diagnóstico del modelo.
- Generación de alarmas de errores, avisos o informaciones en caso de anomalías en el sistema.

Con tal de una correcta accesibilidad de esta interfaz se realizarán diferentes versiones, con herramientas de prototipado para realizar un estudio con *stakeholders*. De esta manera se tiene una información más adecuada de cómo deberían mostrarse esta interfaz, para los usuarios descritos en el punto “3.3. Usuario final”.

4.5. Pruebas en entornos reales

En la última etapa del proyecto, con las fases anteriores completadas, será necesario llevar a cabo la validación en entornos reales. Para esto, se deberá realizar el respaldo de todo el contenido de la máquina de prueba, incluyendo la imagen del sistema operativo, las configuraciones realizadas y la creación de documentación para el manual de uso. Esta documentación servirá tanto para establecer los procesos a seguir en las pruebas realizadas como para futuras instalaciones.

Será práctico probar esta implementación en entornos industriales como el de *Seat* en Martorell, dado su tamaño y proximidad. Además, allí se encuentran una gran cantidad de AGVs del mismo tipo, lo cual es perfecto para probarlo en diferentes máquinas. Esto permitirá analizar posibles problemáticas que no se hayan observado en el entorno de prueba. Durante esta fase, se deberá recopilar de manera sistemática datos e información relevante, y redactar un informe que destaque los inconvenientes encontrados en el nuevo entorno.

5. Definición de los requerimientos

5.1. Requerimientos funcionales

- El modelo ha de ser capaz de aprender de la experiencia.
- El modelo debe controlar tanto la tracción como la dirección del robot.
- El modelo debe elegir acciones con valores mínimos y máximos para proteger la integridad del robot.
- El modelo ha de ser configurable para modificar el comportamiento de la política.
- El modelo debe poder ser habilitado y deshabilitado desde una pantalla integrada en el AGV (HMI).
- El modelo reportará alarmas al HMI en caso de errores o avisos.

5.2. Requerimientos tecnológicos

- La implementación del modelo debe ser compatible en sistemas operativos tanto *Windows* como *Linux*.
- El modelo deberá funcionar a tiempo real, las decisiones que haga deberán tener un lapso casi nulo.
- El modelo deberá funcionar en procesadores *i7* con un mínimo de 8 núcleos.
- El modelo no deberá superar más del 30% los recursos del sistema del robot.

6. Desarrollo

6.1. Puesta a punto del robot

Para este proyecto se ha asignado un AGV de tipo *mouse*. Un robot que se había usado para una instalación de desarrollo para el cliente *Gestamp*[1], de hace unos tres años. Lo primero que se debe hacer es la actualización de los componentes *software*. Todavía mantenía el programa antiguo que ha quedado obsoleto. Los programas internos como el programa del AGV base, el ros, el programa del PLC y las configuraciones de variadores debían pasar a las nuevas versiones.

6.1.1. Actualización componentes software

Uno de los primeros componentes a actualizar es el ordenador industrial. La opción más rápida y sencilla para la actualización es mediante restauración de imagen. Al tener con anterioridad la imagen del sistema operativo de una de los *mouse* de Seat, solo era necesario restaurarlo en el AGV actual. Para ello se hace uso del sistema USB de arranque "*Hiren's Boot*". Este sistema cuenta con diferentes herramientas para realizar mantenimientos o configuraciones en los arranques de ordenadores. Entre otras utilidades, "*AOEMI Backupper*" es el usado tanto para copiar imágenes de sistemas operativos y restaurarlos. Se realiza la restauración entera hacia el ordenador del AGV actual. Con el sistema *Linux* y el sistema ROS actualizado, es necesario actualizar el programa base.

Para actualizar y subir archivos al ordenador, es necesaria una conexión remota. Para esta conexión se usa el programa gratuito "*MobaExterm*". Esta aplicación brinda la posibilidad de conectarse vía SSH (*Secure Shell*), protocolo de conexión remota segura. El programa requiere de la IP (*Internet Protocol*) o el nombre del dispositivo, el nombre de usuario y la contraseña de este. Una vez introducidas y con una conexión en red local tipo inalámbrica o por cable, ya es posible la conexión, como se puede observar en la "Fig 6.1". De esta manera se pueden realizar todo tipo de configuraciones y acciones como si se realizara en el propio ordenador integrado en el robot.

[1] Empresa multinacional dedicada al diseño, desarrollo y fabricación de componentes para el automóvil.

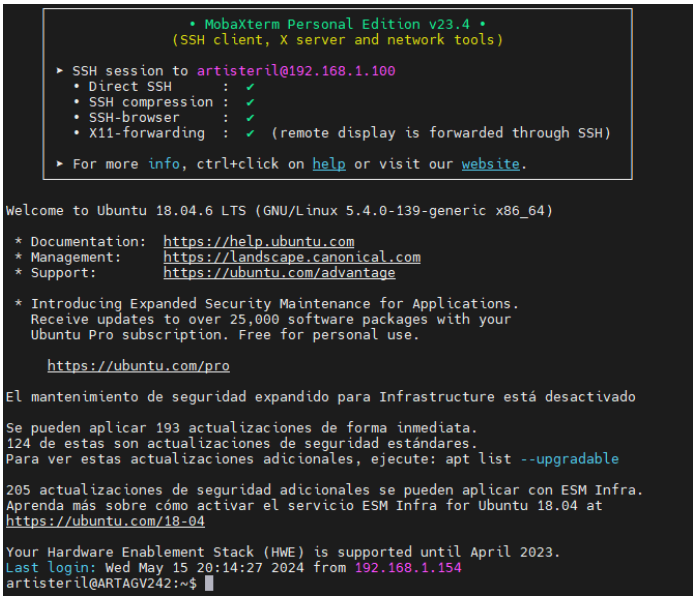

Fig 6.1 Conexión SSH desde la aplicación. Fuente propia.

Se procederá a describir la actualización del programa base del robot. Este recientemente se ha actualizado con la versión de *.Net 8*, por lo tanto, es necesario instalarlo en el sistema. Para la instalación, se requiere ir a la página web oficial para la descarga [47]. Al ser reciente, en el momento de la instalación, no tiene una forma automática de instalarse en *Linux*. Por lo tanto, es necesario descargarse los binarios de la plataforma y añadirlos en la carpeta correspondiente del sistema.

```
artisteril@art-agv-gestamp:~$ sudo chmod 777 -R src
artisteril@art-agv-gestamp:~$ cd /usr/share/dotnet/shared/
artisteril@art-agv-gestamp:/usr/share/dotnet/shared$ ls
Microsoft.AspNetCore.App  Microsoft.NETCore.App
artisteril@art-agv-gestamp:/usr/share/dotnet/shared$ cd Microsoft.AspNetCore.App/
artisteril@art-agv-gestamp:/usr/share/dotnet/shared/Microsoft.AspNetCore.App$ ls
3.0.3  3.1.17
artisteril@art-agv-gestamp:/usr/share/dotnet/shared/Microsoft.AspNetCore.App$ sudo cp -r "/home/artisteril/src/Microsoft.AspNetCore.App/8.0.0/" 8.0.0
artisteril@art-agv-gestamp:/usr/share/dotnet/shared/Microsoft.AspNetCore.App$ ls
3.0.3  3.1.17  8.0.0
artisteril@art-agv-gestamp:/usr/share/dotnet/shared/Microsoft.AspNetCore.App$ cd ..
artisteril@art-agv-gestamp:/usr/share/dotnet/shared$ cd Microsoft.NETCore.App/
artisteril@art-agv-gestamp:/usr/share/dotnet/shared/Microsoft.NETCore.App$ sudo cp -r "/home/artisteril/src/Microsoft.NETCore.App/8.0.0/"
cp: falta el operando archivo de destino después de '/home/artisteril/src/Microsoft.NETCore.App/8.0.0/'
Pruebe 'cp --help' para más información.
artisteril@art-agv-gestamp:/usr/share/dotnet/shared/Microsoft.NETCore.App$ sudo cp -r "/home/artisteril/src/Microsoft.NETCore.App/8.0.0/" 8.0.0
artisteril@art-agv-gestamp:/usr/share/dotnet/shared/Microsoft.NETCore.App$ ls
3.0.3  3.1.17  8.0.0
```

Fig 6.2 Comandos usados para la instalación del paquete .NET 8. Fuente propia.

En la “Fig 6.2” se puede observar el procedimiento de la instalación. Los archivos binarios se alojan en el directorio “*src*”. Es necesario copiar estos binarios al directorio donde el programa base consigue las dependencias, que en este caso es “usr/share/dotnet/shared”. Como se puede observar, además de la versión 8.0 instalada, también se encuentra la antigua, 3.1.

Una vez realizada la actualización de los componentes *software* del ordenador, el siguiente paso es el PLC. Este proceso duró más y dio algún problema, al ser una plataforma cerrada y sin tanto soporte. El objetivo era pasar el programa del PLC, del TIA portal 15.1 a la versión 18. Esta migración se complicó bastante, ya que el componente de comunicación con el variador de los motores dejaba de funcionar con la nueva versión. Este componente, es básicamente, un paquete de funciones de enviar y recibir por un elemento físico puerto

serie. Después de probar diferentes paquetes, proyectos e incluso métodos de actualización del proyecto, se llegó a una solución. La solución consiste en utilizar el programa TIA Portal 18 para abrir el proyecto que se encontraba en la versión 15.1, lo cual automáticamente actualizaba el proyecto a la versión 18. Sin embargo, se siguen utilizando los paquetes de funciones de la versión anterior que ya funcionaban correctamente.

Por último, la actualización de la configuración del variador. Necesaria, ya que recientemente se detectó un fallo en la configuración de los motores y con ella conseguir una mejora del rendimiento. El programa usado es propio de la empresa "*Roboteq*", llamado "*Roborun+*".

6.1.2. Preparación del área de pruebas

Una vez el AGV se encuentra actualizado y preparado para moverse por un entorno, el siguiente paso es preparar esta área. Primero de todo, es necesario realizar el mapeo de la nave donde se encuentra y crear el *layout*, archivo donde se almacenan las trayectorias y movimientos en un plano 2D.

El mapeo se realiza a través de unos paquetes del ROS que gestionan la información de los escáneres láser, procesa los datos que envían y, posteriormente, los transforma en un archivo de imagen a escala de grises. Esta imagen representa los objetos que se va encontrando mientras se mueve por el entorno. Mientras se van realizando vueltas por la nave se puede observar en la "Fig 6.3", el haz de puntos rojos es el escáner láser delantero, que va "dibujando" el mapa donde el color negro son los objetos o paredes más estáticas y a partir que deja de detectar estos objetos se convierten en grises más claros.

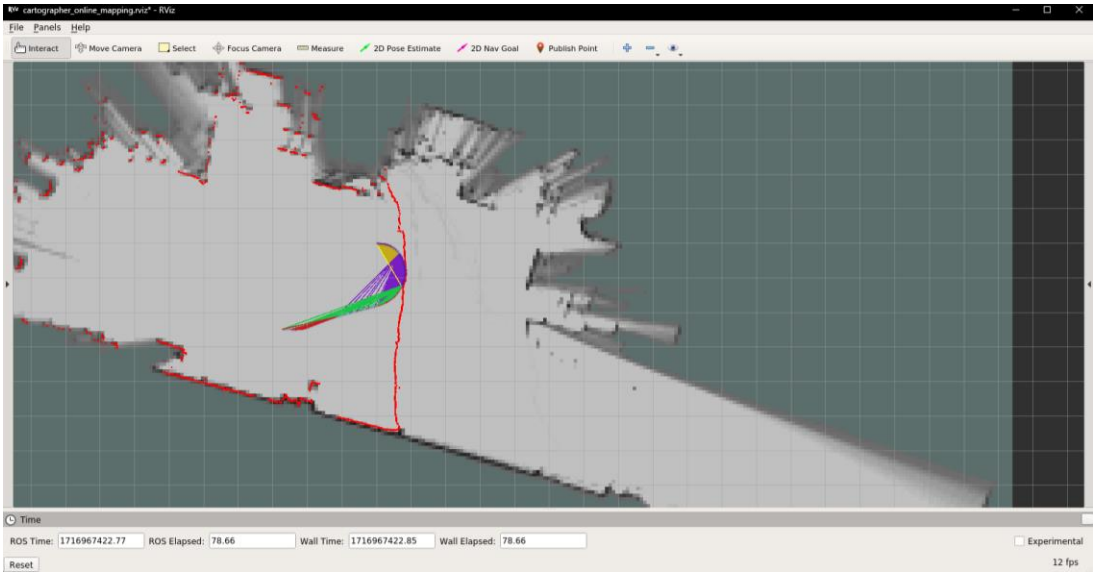

Fig 6.3 Herramienta de visualización mientras se mapea la zona de pruebas. Fuente propia.

Una vez acabado el mapeo, se realiza una serie de comandos para que los archivos generados en el proceso de mapeo se transformen en una imagen PNG (Portable Network Graphics). En la siguiente "Fig 6.4" ya se obtiene el mapa de la nave donde se realizarán las pruebas. Para que la máquina pueda moverse por el entorno en automático es necesaria la definición de sus trayectorias y para ello también es necesario el *layout*.

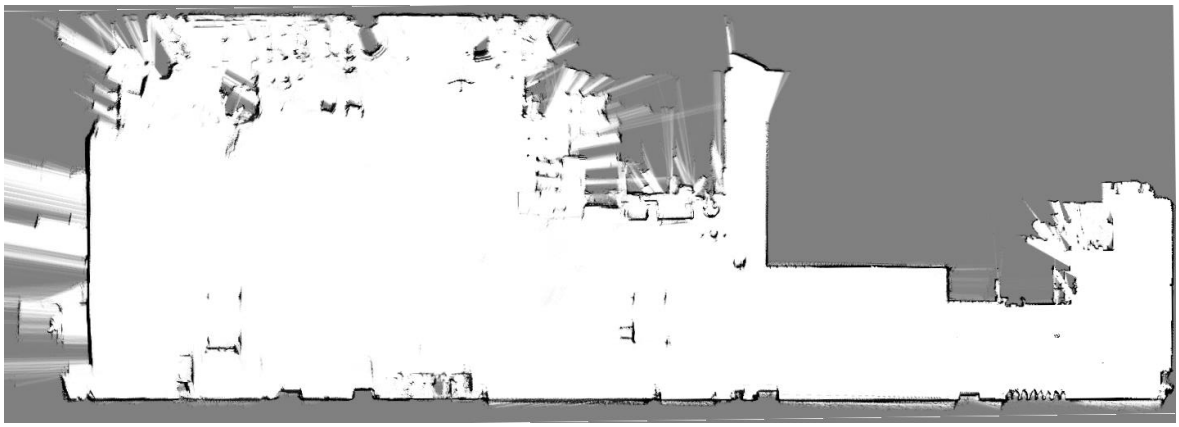

Fig 6.4 Mapeo realizado para las pruebas. Fuente propia.

Como ya se ha mencionado, el *layout* es un archivo guardado en formato JSON (JavaScript Object Notation), usado para el guardado y acceso de datos. En este caso es usado para crear, acceder y modificar los diferentes objetos que se usan para definir la trayectoria del AGV, como pueden ser, segmentos, curvas, estaciones, *links* (enlaces entres estaciones). Este archivo se crea y se modifica de forma visual con un programa hecho por la empresa, llamado "ArtisterilCAD". Como se puede observar en la "Fig 6.5", se encuentra ya un *layout* creado con sus respectivos segmentos y estaciones. En la parte inferior se pueden ver los *links*, son los enlaces de las estaciones, en una definición simple son los puntos "a" y "b" que puede ir el robot.

Como también se puede ver, está referenciado con el mapa realizado previamente, que se encuentra en el fondo de la "Fig 6.5". Una vez completados estos preparativos del área de pruebas ya se pueden comenzar a realizar las primeras pruebas de rendimiento en automático.

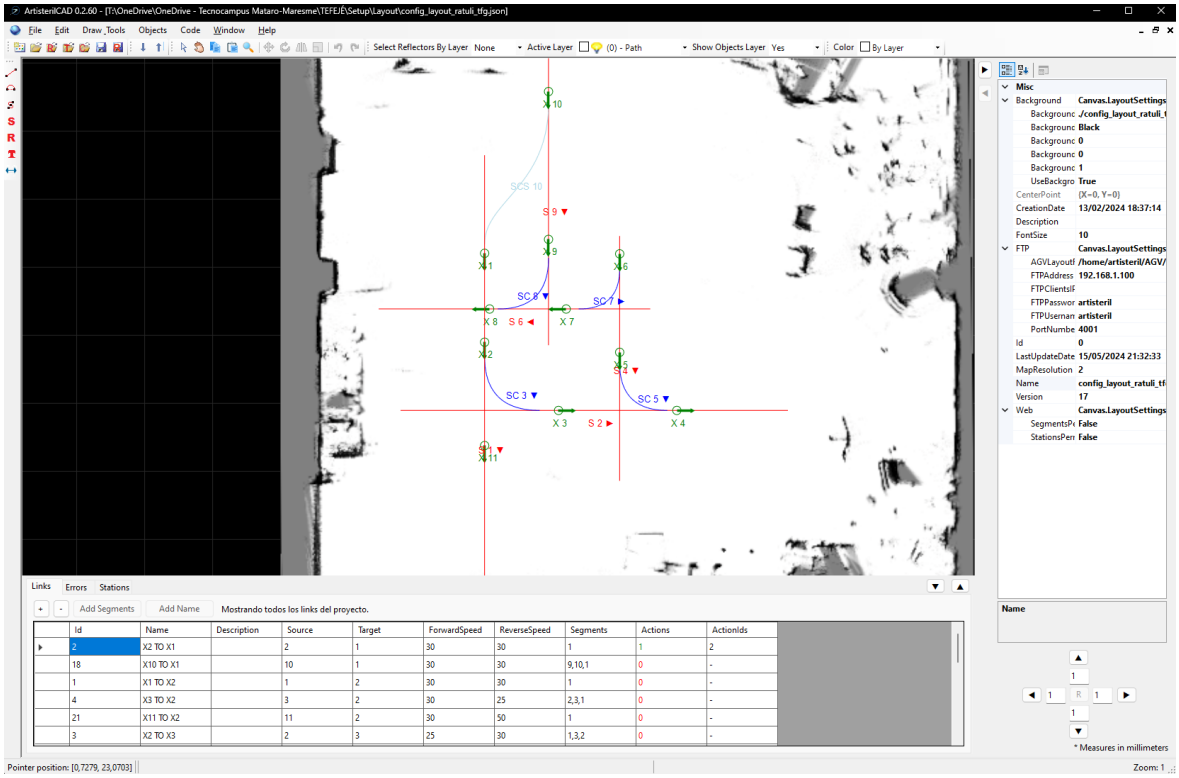

Fig 6.5 Layout desde la herramienta de dibujo ArtisterilCAD. Fuente propia.

6.1.3. Instalación Python y dependencias

Cronológicamente este proceso se ha realizado en un momento más madurado del desarrollo del modelo, cuando se comenzó a realizar las pruebas en la máquina real. Aún así está dentro de la puesta a punto del robot para que pueda usar el programa en Python realizado.

Para ello es necesaria en la instalación de Python en primera instancia. En este caso en concreto, el programa está escrito en la versión 3.11.0 y se instalará en esta versión para evitar posibles incompatibilidades.

El proceso paso a paso se realiza en la conexión remota SSH, mencionada anteriormente, introduciendo los siguientes comandos:

- La instalación las dependencias para compilar el código fuente de Python: "*sudo apt install -y build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev libssl-dev libreadline-dev libffi-dev libsqlite3-dev wget libbz2-dev*"
- La descarga del código fuente de Python 3.11.0 desde la página oficial y la extracción del archivo comprimido, respectivamente:

"*wget https://www.python.org/ftp/python/3.11.0/Python-3.11.0.tgz*", "*tar -xf Python-3.11.0.tgz*"

- Se debe entrar a la carpeta extraída, realizar una configuración de compilación y por último compilar el código con los siguientes comandos respectivamente:

"*cd Python-3.11.0*", "*./configure --enable-optimizations*", "*make -j $(nproc)*"

- Finalmente instalar Python con el comando: "*sudo make altinstall*"
- Verificar que se ha instalado con: "*python3.11 --version*"

```
artisteril@ARTAGV242:~$ python3.11 --version
Python 3.11.0
```

Fig 6.6 Verificación de Python 3.11.0 instalado. Fuente propia.

Una vez instalado Python en el ordenador integrado del AGV, también es necesaria la instalación de las dependencias usadas en el programa del modelo RL, entre otras, *PyTorch*. Para ello se usará el método de entornos virtuales, que se definen sencillamente como espacios aislados en el ordenador para poder realizar la instalación de las dependencias de uno o varios programas. En este caso se realiza la instalación paso a paso del entorno virtual para el programa del modelo de la siguiente manera.

- Realizar la instalación de "*virtualenv*", el paquete encargado para crear entornos virtuales. Se instala con el comando: "*pip install virtualenv*"
- Crear un nuevo entorno virtual con el nombre deseado, en este caso "modelvenv". Con el comando "*virtualenv modelvenv*" se crea. En este caso en concreto, el ordenador ya tenía otra versión de Python, más específicamente la 2.6. Por lo tanto se debe concretar en cual versión de Python crear el entorno, con el siguiente comando:
  "*virtualenv -p /usr/local/bin/python3.11 modelvenv*"
- Una vez creado el entorno virtual, para utilizarlo, se debe activar:
  "*source modelvenv/bin/activate*"
  En la "Fig 6.7" se puede confirmar que está activado ya que está marcado entre paréntesis el entorno activo.

Fig 6.7 Entorno virtual creado correctamente. Fuente propia.

- A partir de aquí, ya se instalan las librerías necesarias para los programas que se usen dentro del entorno. En el caso en concreto del modelo, se ha creado un archivo de texto "*requirements.txt*" donde se encuentran los paquetes a instalar y sus versiones, entre ellos el *PyTorch*. Se instalan con el siguiente comando:

  "*pip install -r requirements.txt*"

Hechas las instalaciones necesarias para el programa del modelo con el *framework* de Python se puede realizar las pruebas desde la máquina real con el programa del modelo de RL.

6.2. Desarrollo de los KPI

En esta sección se detallará la manera en que se obtiene e interpreta la información del AGV para el desarrollo de los indicadores de rendimiento, KPI. De esta manera extraer saber si el modelo está realizando correctamente el objetivo principal.

6.2.1. Precisión del modelo

Como se ha explicado en el punto "3.4.1. Precisión del modelo", se realizan dos medidas: las desviación lateral mientras el AGV está navegando, la diferencia entre la estación virtual y la parada final de la máquina.

Para ello, se ha implementado una parte del código en el programa base, específicamente en el módulo "*Navigation*", que se encarga de gestionar la información de la localización de la máquina y calcular diversas medidas respecto a la trayectoria, entre las cuales se encuentra la desviación lateral.

Para extraer una medida fidedigna, se ha implementado un guardado de este dato cada ciertos milímetros que el AGV navega. Esta distancia es configurable mediante un parámetro llamado "*MmToRecordNavigationMeditions*" en el archivo JSON de configuración, denominado "*config_navigation.json*". Este archivo contiene todos los parámetros que influyen en la navegación de la máquina. La distancia predeterminada es 500 mm; sin embargo, para evitar que las medidas se tomen en las mismas partes del circuito repetidamente, se ha añadido un pequeño desplazamiento aleatorio que incrementa o disminuye esta cifra en un pequeño ratio.

Una vez que estos datos se guardan temporalmente desde el programa, cada vez que el AGV pasa a modo manual o transcurre un tiempo considerable, estas mediciones se almacenan en una base de datos interna. Esta base de datos está implementada con el paquete "LiteDB" [48], una base de datos simple no relacional enfocada al desarrollo en .NET.

Por otra parte, las mediciones de parada en los destinos del AGV se han programado de la siguiente manera: en la misma clase "*Navigation*" se lanza un evento cuando el AGV llega a una coordenada de final de *link*. Cuando este evento se recibe, se verifica si es la coordenada del destino final y se llama a una función que guarda la posición actual del robot y la posición virtual a la cual debería llegar.

Para una visión más genérica y sencilla, se puede observar la "Fig 6.8" un diagrama donde se muestran únicamente las clases encargadas de este KPI. La tarea "*Navigate*" gestiona gran parte de la navegación y contiene el código previamente explicado para guardar las desviaciones laterales, llamando a la clase "*DataNavigation*", que tiene acceso a la base de datos para almacenar estos datos. De la misma manera, se observa que cuando ocurre el evento de llegada a una estación, se llama posteriormente al guardado del punto de parada si esta estación es el destino.

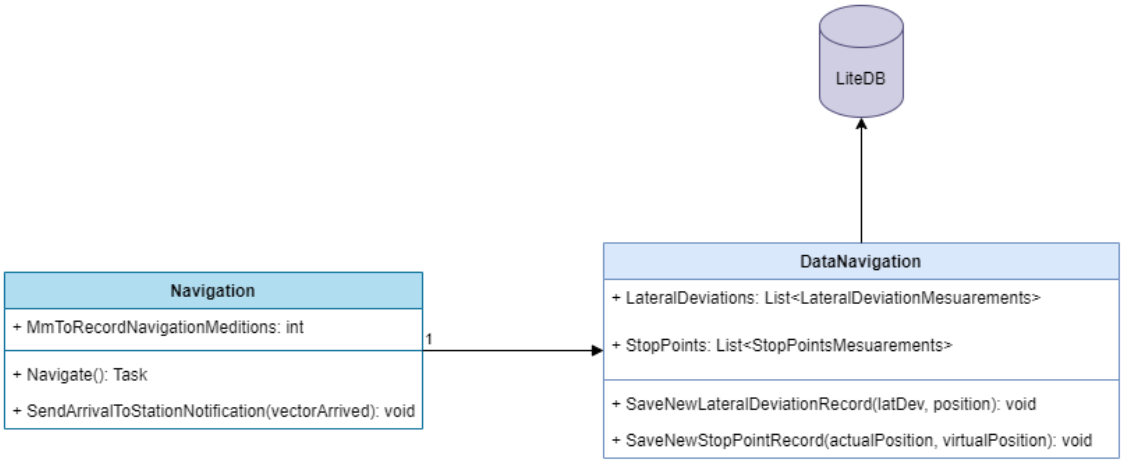

Fig 6.8 Diagrama de clases simple donde se muestra la implementación de los datos de la navegación. Fuente propia.

Estos datos no tienen valor si no se procesan adecuadamente. Por lo tanto, se extraen desde la misma máquina, convirtiendo la información almacenada en la base de datos a archivos CSV (valores separados por comas). Estos archivos se procesan en el programa *Power BI* para su mejor visualización y para facilitar la comparación de las pruebas realizadas.

6.2.2. Tiempo de convergencia

Este KPI es manejado exclusivamente por el programa Python del modelo RL. La forma de medirlo es similar a cualquier otro proyecto de aprendizaje profundo, al implementar el algoritmo DQN. Se mide a partir del tiempo en el que los valores de pérdida convergen, es decir, cuando el modelo se estabiliza al aprender durante un periodo de tiempo.

La diferencia radica en que únicamente se puede medir el valor de pérdida en la etapa de entrenamiento, ya que en la validación no se dispone de un "respaldo" de un valor correcto. En la “Fig 6.9” se puede ver un ejemplo de gráfica. En esta gráfica, el valor de pérdida se encuentra en el eje y, mientras que el eje x representa las etapas. Lo habitual en este tipo de gráficas es que los valores tiendan hacia abajo, lo que indica que el valor de pérdida disminuye, significando que el error del modelo es menor.

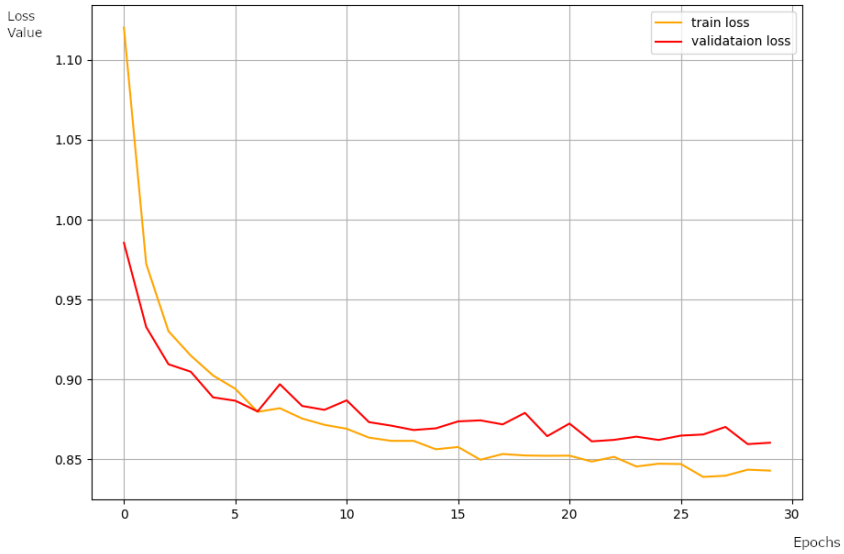

Fig 6.9 Gráfica del valor de pérdida, proyecto propio de clasificación de género en imágenes.

Fuente propia.

En este proyecto se utiliza el paquete de herramientas *Tensorboard*, empleado para almacenar datos y visualizarlos de forma sencilla e intuitiva. Se guarda el valor de pérdida de *Q* en los momentos de aprendizaje del modelo, que corresponden a los periodos en los que la máquina no se está moviendo. Además de almacenar el valor de recompensa que devuelve el entorno del programa RL. A partir de estas gráficas se determinará en cuántas etapas el modelo ha logrado “aprender” y el tiempo total que ha tardado en hacerlo.

6.2.3. Tiempo entre predicciones

Este indicador de rendimiento se obtiene a partir de la diferencia de tiempo entre predicciones. La implementación se realiza de la siguiente manera: en el momento en que se establece la comunicación entre el programa base y el modelo RL, si se cumplen las condiciones para empezar a predecir, explicadas con más detallen en el punto “6.3.1”, se recogen los datos de recepción entre predicciones del modelo.

El KPI se obtiene de manera muy directa desde el momento en que se envía la petición de predicción, juntamente con el estado del AGV y el tiempo en que llega la predicción. Dado que se trata de unidades de tiempo muy pequeñas y de muchas medidas a lo largo del tiempo, se recogen un elevado número de medidas, concretamente quinientas. Una vez obtenidas, se calcula la media, la mediana, la varianza, el mínimo y el máximo de estas diferencias de tiempo. Finalmente, estos valores se almacenan en la base de datos para obtener el archivo CSV y realizar gráficas en *Power BI*.

6.2.4. Uso de los recursos

Por último, se encuentra la implementación del uso de los recursos del sistema. Uno de los objetivos del es el que no supere cierto umbral del porcentaje de uso del sistema, como se explicó en el punto “3.4.4”. Este uso de sistema se medirá únicamente en el programa del modelo RL, ya que en el programa base no tiene sentido al utilizarse como pasarela de comunicación y su porcentaje de uso de CPU es casi inapreciable.

Por ello se ha implementado la misma técnica que en el KPI del tiempo de convergencia. Con la herramienta *TensorBoard* se almacenan los datos, el porcentaje de la CPU y los megabytes (MB) de memoria utilizados por el programa. Se ha utilizado el método de lanzamiento de eventos; cada dos segundos, un evento se dispara y activa una función de guardado de los datos previamente mencionados. De esta manera, los datos recopilados se pueden visualizar posteriormente en una gráfica para identificar en qué puntos el programa ha sido más demandante.

6.3. Desarrollo del programa

En esta sección se explica el diseño, implementación y desarrollo del programa realizado en Python para el modelo RL. Así mismo de las modificaciones del programa base del en C#, para el envío y recepción de los datos necesarios para lograr los objetivos.

Durante esta sección se referirá al programa base del AGV como *AGVBase* y al programa del modelo RL como *RLModel*, con la finalidad de economizar la escritura de las palabras.

6.3.1. Protocolo de comunicación

Como ya se explicó en el punto “2.1.4”, el *AGVBase* controla una gran parte del AGV. Este programa está escrito en C#, mientras que el *RLModel* se ha desarrollado en Python, lo que implica que no existe una forma directa de compartir datos o variables del sistema entre ellos. Por ello, se ha decidido establecer la comunicación entre los dos elementos mediante el protocolo UDP.

La elección de UDP se debe principalmente a su alta escalabilidad, dado que es un protocolo de comunicación dentro de la capa *Ethernet/IP*. Este enfoque ofrece una solución muy flexible y permite, en el futuro, la comunicación en red local o externa con otro ordenador, una máquina virtualizada o incluso un contenedor de virtualización de aplicaciones, como *Docker*. Además, otro punto fuerte es su alta velocidad, al ser un protocolo no orientado a la conexión. Esto permite una transmisión que minimiza cualquier tipo de retraso, maximizando así el tiempo entre predicciones del modelo.

Una vez definida la base de la conexión entre los programas, se procederá a explicar el diseño del protocolo comunicación, es decir, la manera en que el *AGVBase* informa al *RLModel* y viceversa. En el siguiente diagrama en la “Fig 6.10” se puede entender de manera más visual el diseño de esta comunicación.

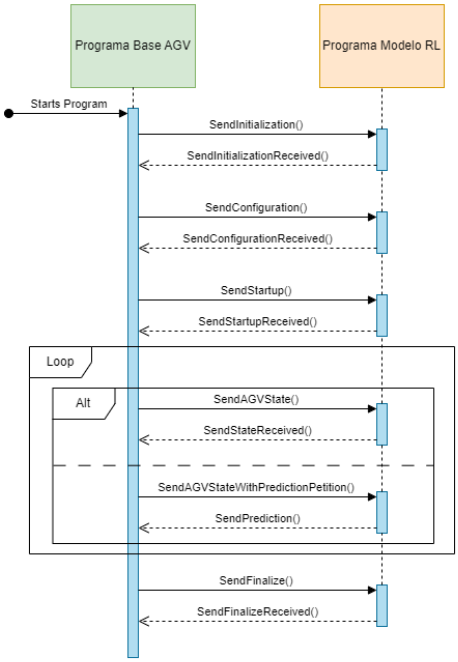

Fig 6.10 Diagrama del flujo de la comunicación, entre el programa de AGV y el programa del modelo RL. Fuente propia.

Al iniciarse los dos programas, el que lidera la inicialización es el *AGVBase*, que envía el mensaje de inicialización de la comunicación. Si el *RLModel* no está iniciado, *AGVBase* envía el mensaje de inicialización a intervalos regulares hasta obtener una respuesta. Una vez *RLModel* responde, descrito en el diagrama mencionado anteriormente. *AGVBase* envía la configuración con toda la información que *RLModel* necesita para instanciar el entorno, el agente y el modelo de red neuronal.

Una vez recibida la configuración, *AGVBase* decide cuando debe comenzar el intercambio de estado-acción que el algoritmo de RL requiere para funcionar. Este ciclo comienza cuando el AGV se encuentra en automático; *AGVBase* envia el estado del AGV para que *RLModel* tenga el conocimiento previo de su ubicación. Una vez el AGV comience a moverse con normalidad, se inician las peticiones de predicción, es decir, las acciones que debe tomar el *RLModel*.

En el momento que el AGV pase a estado manual, se finaliza el ciclo de estado-acción previamente explicado y *AGVBase* envía la señal de finalización. En este punto, *RLModel* confirma de la recepción de la finalización y comienza su entrenamiento, si es necesario. Este entrenamiento se explicará en mayor detalle en el punto “6.3.4”.

6.3.2. Arquitectura utilizada

Se ha explicado cómo se comunican los programas entre sí. Durante esta sección se explicará diseño del software utilizado, ya que es necesario el uso de patrones para que la funcionalidad sea escalable y entendible. En este caso se ha usado la arquitectura hexagonal [49] para realizar la implementación del protocolo de comunicación en el *AGVBase*.

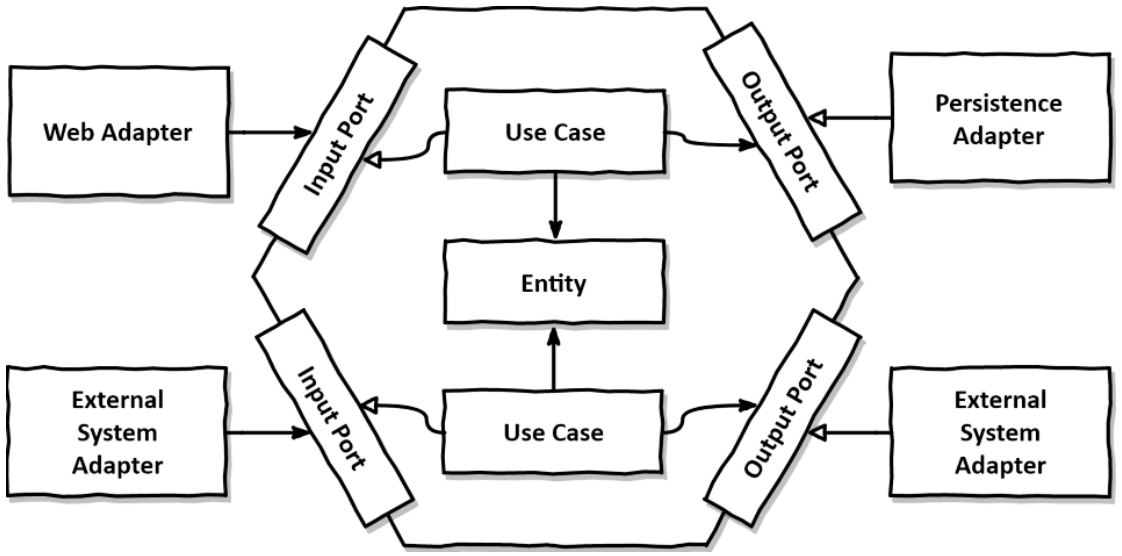

Fig 6.11 Esquema de la arquitectura hexagonal

Esta arquitectura permite aislar los diferentes componentes del programa para obtener una mayor escalabilidad y flexibilidad. Como se puede observar en el esquema de la "Fig 6.11" se diferencia el elemento entidad, los casos de uso, los puertos tantos de salida como entrada y los adaptadores externos. Esto se traduce en que el código realizado se debe hacer basándose en esta arquitectura.

Lo principal es identificar los diferentes módulos, "*ReinforcementLearningModel*" es la entidad base de esta arquitectura. Por otra parte, el caso de uso es el protocolo de comunicación implementada, los puertos de entrada son los mensajes recibidos por parte del *RLModel* y los puertos de salida los mensajes que se envía desde *AGVBase*. Los adaptadores externos son las interfaces de conexión ya comentadas con el protocolo UDP. Finalmente, el diseño de la arquitectura está representada en el siguiente esquema "Fig 6.12" basándose en el anterior.

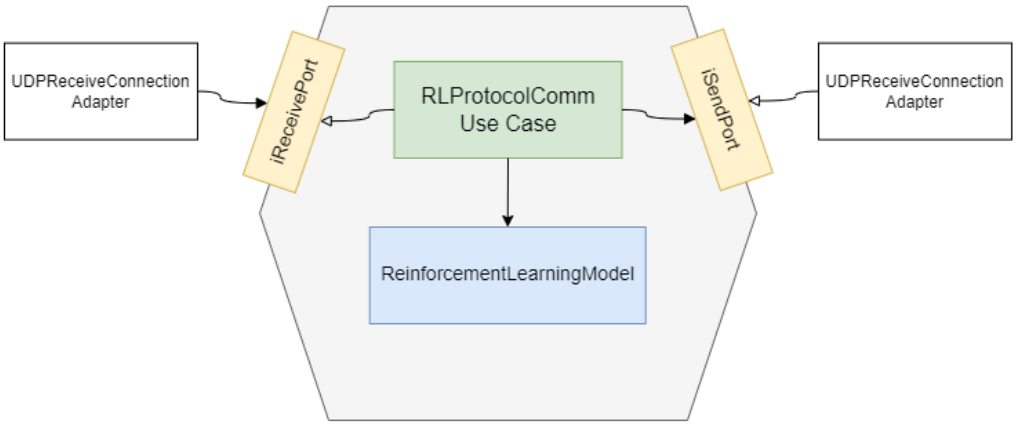

Fig 6.12 Esquema de la arquitectura hexagonal aplicado en el software. Fuente propia.

Esta arquitectura se ha implementado con relativa facilidad en el software de *AGVBase*, al estar desarrollado en C#, se ha seguido las prácticas de los ejemplos en JAVA al ser lenguajes altamente parecidos y con capacidades similares. A diferencia del software *RLModel* se ha intentado seguir la misma arquitectura, aunque Python, al no tener interfaces ni nada parecido, se ha implementado con el método de inyección de dependencias. Funcionalmente es muy parecido, pero le quita la importancia del aislamiento entre módulos.

6.3.3. Intercambio de datos

En este punto se describirá las tramas usadas en la comunicación de los programas desarrollados. Ya se ha explicado el flujo del protocolo de comunicación UDP, pero no los datos e información transmitidos, además del formato utilizado.

Para esto se ha decidido usar el formato JSON, que ya se emplea para los archivos de configuración y está estandarizado en *AGVBase*. Implementarlo en *RLModel* es sencillo

gracias al paquete de herramientas "*json*". Esta herramienta es capaz de transformar las tramas recibidas por el canal de comunicación en objetos más manejables, para su uso en los módulos requeridos por *PyTorch* para el modelo de aprendizaje por refuerzo. De la misma forma que se transforman los mensajes a objetos, se realiza el proceso inverso para el envío de las predicciones realizadas por el modelo.

Estas transformaciones son posibles gracias a la estandarización de los objetos. A continuación, se van a enumerar estos objetos junto con su diagrama de clases. En el mensaje de configuración se envía el objeto "*ConfigurationArgsModel*", como se muestra en la "Fig 6.13", donde se definen los diferentes parámetros que *RLModel* requiere. Este objeto se envía mediante el método "*SendConfiguration*".

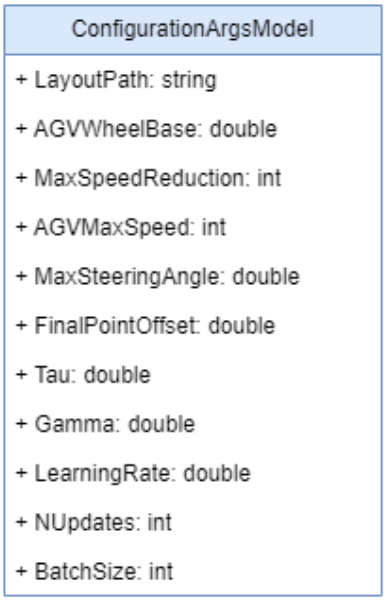

Fig 6.13 Diagrama del objeto ConfigurationArgsModel. Fuente propia.

Siguiendo el flujo de mensajes del protocolo, se encuentra el método "*SendStartup*", encargado de enviar el objeto "*StartupArgsModel*" definido en la "Fig 6.14". En este diagrama se puede observar el envío de dos parámetro: primero, la lista de los identificadores de los enlaces que el AGV debe recorrer en su trayectoria actual, para que el modelo tenga conocimiento de esta; y segundo, el estado actual de la máquina, para poder inicializar el entorno del elemento de RL.

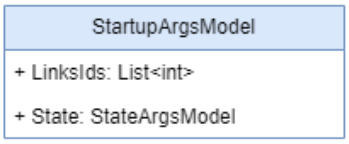

Fig 6.14 Diagrama del objeto StartupArgsModel. Fuente propia.

El parámeto estado del AGV en el anterior objeto, como se puede ver, es otro objeto llamado "*StateArgsModel*". Donde, una vez más, en el diagrama "Fig 6.15" se pueden ver las diferentes variables que se envían. Este objeto se envía para informar el estado de la máquina y también usa el parámetro "*ComputePrediction*" para realizar la petición de predicción del modelo. Si *RLModel* recibe esta variable como verdadera, empieza el proceso de generar la acción.

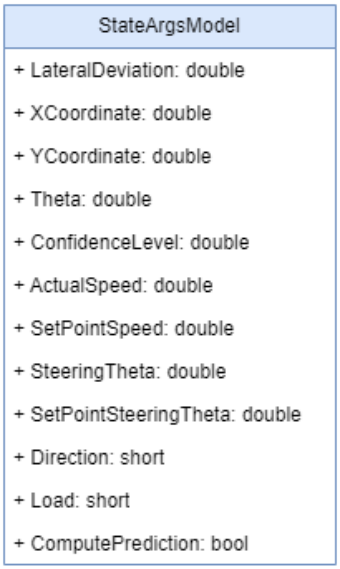

Fig 6.15 Diagrama del objeto StateArgsModel. Fuente propia.

En cuanto al objeto de acción que *RLModel* envía, se define en la siguiente "Fig 6.16". El diagrama de clase del objeto "*ModelAction*" contiene la información de la decisión que ha tomado el modelo, incluyendo la reducción de velocidad (que finalmente no se ha utilizado) y el ángulo de dirección de la rueda. Además, cuenta con parámetros para el seguimiento de las predicciones, como el instante en el que se ha recibido, si la acción se ha utilizado o no (lo cual se explicará con más detalle en el punto "6.3.5") y la acción realmente utilizada.

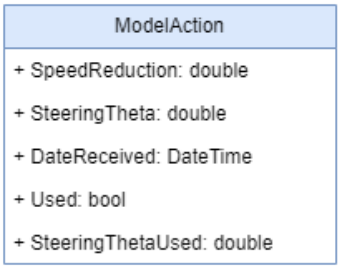

Fig 6.16 Diagrama del objeto ModelAction. Fuente propia.

Para finalizar con la enumeración de los objetos usados para la transmisión de los datos, se describe el método encargado de finalizar la transmisión estado-acción del modelo, "*SendFinalize*". El objeto se denomina "*FinalizeArgsModel*", definido en la "Fig 6.17", y este informa sobre la razón de la finalización. En el diagrama también se encuentra el "*Enum*", estructura de datos usada en C#, donde se enumeran las diferentes razones para dicha finalización. Esto está diseñado de esta manera porque, según la razón que reciba, *RLModel* decide comenzar el proceso de aprendizaje o no.

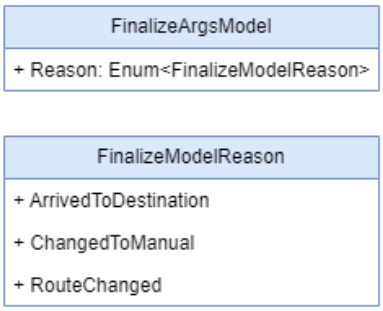

Fig 6.17 Diagrama del objeto FinalizeArgsModel, juntamente con el *Enum* de las razones. Fuente propia.

Este diseño de objetos que se convierten a formato JSON y viceversa permite que el desarrollo sea flexible, ya que se pueden añadir, modificar o eliminar parámetros sin necesidad de realizar múltiples cambios en el código. Además, para el protocolo de comunicación UDP, este formato es transparente, ya que, al transformarse en cadenas de texto, se envían y reciben sin consumir muchos recursos.

6.3.4. Modelo RL

En esta sección se explicará el núcleo del proyecto. El modelo desarrollado en el lenguaje *Python*, utilizando en gran medida el paquete de herramientas *PyTorch*. Esta sección se distribuye de manera que se revisan los elementos de RL, ya explicados en el punto "2.2.6". Se detalla cómo interactúan entre ellos y cuál es la función de cada uno.

6.3.4.1. Entorno

El entorno es uno de los pilares en RL, ya que es el encargado de recibir la acción generada por el agente, realizar el cambio de estado y calcular el valor de la señal de recompensa. En este caso en concreto, el entorno se ha implementado con el módulo *EnvBase* de la librería "*torchrl.envs*". Esta herramienta usa los mismos métodos base y variables de la API *gym*, por lo que se debe prestar especial atención en los métodos *reset*, *step*, además de las variables *observation_spec*, *state_spec* y *action_spec*.

El parámetro *observation_spec* hace referencia al entorno físico del agente, en donde se mueve y el dominio que comporta este espacio. En este caso específico, esta variable está definida como un *CompositeSpec*, un objeto que es una composición de tensores, estructura de datos explicada en el anterior punto "2.3.1.1".

Los tensores que corresponden a esta composición son vectores con el tipo de dato "*float64*" que representan la trayectoria. De manera que están almacenadas las coordenadas *x*, las coordenadas *y*, el ángulo teórico del AGV, el ángulo teórico de la rueda de dirección del AGV y, por último, la velocidad teórica en ese punto del trayecto. Se puede observar una representación más visual en la "Fig 6.18", de puntos finitos de los tensores de las coordenadas *x* e *y*. Que a su vez representa la trayectoria del AGV según el *layout*, en la "Fig 6.5", que se ha enviado en la configuración *AGVBase*.

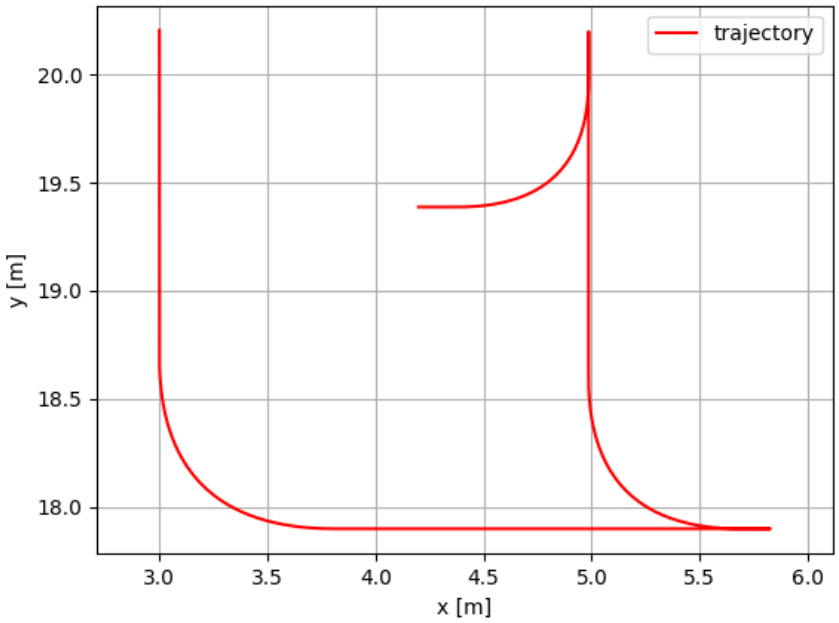

Fig 6.18 Representación visual de los puntos del trayecto que debe trazar el AGV. Fuente propia.

El método de transformación de la ruta del *layout* a la trayectoria ya mencionada es código protegido por la empresa, pero, como breve explicación, se trata de la clase "*TrajectoryGenerator*". Esta clase es responsable de leer el archivo JSON del *layout* y, según

la configuración de la máquina, realizar cálculos para convertirlo en los vectores de puntos referenciados anteriormente.

Una vez definido *observation_spec*, se debe mencionar la relación entre esta composición y el denominado *state_spec*. Este último, en el entorno, representa al estado del agente, en el caso del proyecto, el AGV. En la implementación del entorno, *state_spec* también se ha definido dentro del *observation_spec*, siguiendo el procedimiento del ejemplo [41]. Dado que el ejemplo de *Cartpole* es un entorno sin estado, en el cual el propio entorno es el agente que debe mantenerse recto, esto implica que cada paso de información del estado también incluye la información de la trayectoria, que no debería cambiar hasta que no se realice un cambio de trayectoria. Esto resulta en una menor optimización de los recursos y representa un punto de mejora a resolver.

Como último parámetro se encuentra el *action_spec*, que una vez más es un *CompositeSpec* que representa el dominio de posibles valores correspondiente a la acción del agente. En este proyecto, *action_spec* se ha diseñado como un tensor de tipo "*float64*" correspondiente al valor de consigna del ángulo del motor de giro. Además, se ha dejado preparada la funcionalidad de reducción de velocidad, ya que puede ser relevante disminuir la velocidad si con ello se mejora la precisión. Sin embargo, esta funcionalidad se ha descartado por motivos de complejidad extra. Los valores máximos y mínimos de estas variables están definidos, nuevamente, por la configuración recibida, vista en el diagrama "Fig 6.13".

Por último, se presentan las definiciones de los métodos *reset* y *step*. El método *reset* tiene la responsabilidad de inicializar todas las variables anteriormente mencionadas y devolver el estado actual del entorno. Por otro lado, el método *step* recibe la acción y el estado actual de la máquina, devolviendo el valor de la recompensa. La función de recompensa implementada es la siguiente:

$$r = \exp\left(6.5 \cdot l^2 + 3.1 \cdot \Delta h^2 + 1.5 \cdot \Delta\theta^2 + 0.05 \cdot {\Delta\theta_t}^2\right) \quad (6)$$

La variable *l* representa la desviación lateral de la máquina, mientras que Δh orresponde a la diferencia de ángulo de la máquina con respecto al trayecto establecido. La variable $\Delta\theta$ denota la diferencia entre la consigna del ángulo de la rueda de dirección y el ángulo actual de esta. Finalmente, $\Delta\theta_t$ es la diferencia entre la consigna del ángulo de la rueda de dirección y su ángulo teórico.

Como se puede observar en la fórmula (6), todos los componentes están elevados al cuadrado, eliminando así la posibilidad de obtener valores negativos. Además, cada uno de los componentes tiene una ponderación específica para normalizar las medidas y asegurar que ciertos componentes, como *l*, tengan mayor relevancia que otros. Finalmente, todos estos componentes se encuentran dentro de una función exponencial, lo que permite que la recompensa se condense en valores en el intervalo [0, 1]. Esto contribuye a la normalización de los valores de recompensa, mejorando así el proceso de aprendizaje del modelo.

6.3.4.2. Agente

El agente RL, es el responsable de inicializar la política y calcular los valores de la función. En este caso, al tratarse de una política implementada mediante aprendizaje profundo (DL), el agente se encarga de inicializar las redes neuronales (la red local y la red objetivo), así como el optimizador. Además, se ocupa de almacenar los pasos (s, a, r, s') para el proceso de aprendizaje, es decir, reajustar los valores de las redes neuronales.

Este agente contiene los hiperparámetros para la política y el optimizador, como el *learning rate*, el hiperparámetro responsable de la magnitud de la actualización de los pesos en la red neuronal durante el proceso de aprendizaje. Otros hiperparámetros relevantes, configurables a través de ficheros JSON, son los siguientes:

- *Tau*: la tasa que controla la actualización de los parámetros entre la red objetivo y la red local.
- Gamma: determina la relevancia de las recompensas futuras en comparación con las recompensas inmediatas.
- *NUpdates* (número de actualizaciones): el número total de procesos de aprendizaje a realizar durante el método de aprendizaje.
- *Batch Zize:* el tamaño del lote de muestras utilizadas en un proceso de aprendizaje, que afecta a la estabilidad y velocidad del entrenamiento.

En los anexos se encuentra los parámetros por defecto usados durante la prueba.

En el momento de la inicialización, también se encarga de revisar si existen modelos previamente guardados y cargarlos utilizando la función "*torch.load*". De esta manera, se asegura que los procesos de aprendizaje ya realizados no se pierdan al apagar el AGV o el programa.

El agente contiene además un módulo denominado "*PrioritizedReplay*", encargado de almacenar en memoria las experiencias (s, a, r, s') mencionadas anteriormente y, al inicio del proceso de aprendizaje, asignar una prioridad a los pasos recientemente guardados. Esto permite lograr un aprendizaje más eficiente y fiable.

Como se ha comentado, el agente contiene la política del modelo, implementada mediante una red neuronal simple (explicada en el siguiente apartado). Esta red neuronal se encarga de realizar el proceso de predicción mediante el método *act* el cual recibe el estado del entorno, realiza el proceso de *forward propagation* y determina la acción a enviar al AGV.

El método final es el aprendizaje, denominado *learn_per*. Este método utiliza un lote de experiencias $(s, a, r, s', \text{indicadores de finalización}, \text{índices y pesos})$ para calcular los valores objetivo y los valores esperados del modelo local. Calcula el error temporal diferencial (TD error) y la pérdida ponderada, retropropaga esta pérdida y actualiza los parámetros del modelo. Además, realiza una actualización suave de la red objetivo y ajusta

las prioridades en la memoria de experiencias mencionada anteriormente. Finalmente, devuelve el valor de la pérdida, valores que se utilizan en el KPI "Tiempo de convergencia".

6.3.4.3. Red Neuronal

La implementación de la red neuronal se ha reutilizado del programa de ejemplo discutido en el punto "2.4.2" [43]. Se encontraban disponibles dos versiones: una con una única capa oculta ("*hidden layer*") y otra versión denominada "*DeepNAF*" la cual está diseñada con tres capas adicionales. A pesar de tener un mayor número de capas ocultas, estas siguen siendo capas lineales simples con una función de activación definida como "*Batch Normalitzation*". En la "Fig 6.19" se muestra el diagrama de las diferentes propiedades de la red neuronal.

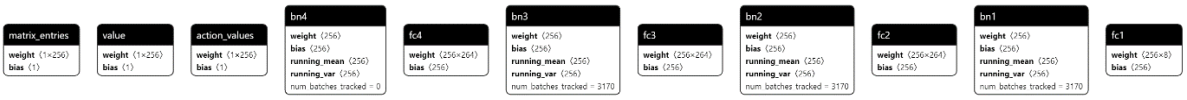

Fig 6.19 Diagrama de las propiedades de la red neuronal. Fuente propia con la herramienta *Netron* [50].

Es importante mencionar que tanto la red local como la red objetivo tienen el mismo diseño. Aspectos relevantes del diseño incluyen que cada capa tiene un tamaño de 256 neuronas, lo que implica que cada capa contiene un total de 256 entradas más la propia entrada del estado.

Este estado de entrada está definido como un "*TensorDict*", que en términos simples es un tensor que contiene una estructura de datos de tipo diccionario. Esta estructura incluye el nombre de la variable y su valor correspondiente. El diseño implementado para la entrada de las redes neuronales se ilustra en el siguiente diagrama, "Fig 6.20".

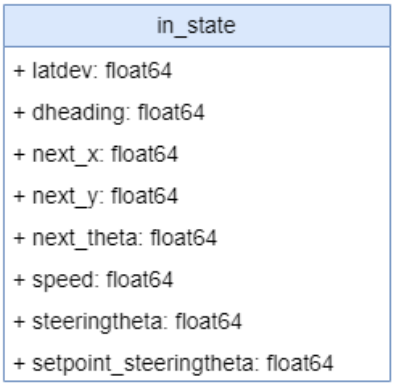

Fig 6.20 Diagrama de los datos de entrada a la red neuronal. Fuente propia.

Estos parámetros de estado de entrada se han modificado progresivamente y es probable que continúen siendo investigados para una mejor optimización en la recepción de información por parte de la red neuronal. De la misma manera que se puede mejorar la información que se le proporciona a la red, se puede mejorar la propia red neuronal, ya que actualmente es

bastante simple y requiere una forma de tener una información temporal de los datos recibidos. Una línea de investigación reciente ha sido sobre las capas denominadas LSTM (*Long Short Term Memory*), las cuales son capaces de almacenar un tipo de memoria y decidir si utilizar o no la información recibida.

6.3.4.4. Orquestador

Para finalizar la sección del desarrollo del modelo RL, se presenta una breve explicación de un componente denominado "orquestador". Este componente actúa como una clase puente que gestiona los eventos de los mensajes recibidos, instancia el generador de la trayectoria cada vez que hay un cambio de ruta, instancia al agente RL y envía la información de vuelta a *AGVBase*.

El orquestador tiene la responsabilidad de recibir la información del estado por parte de *AGVBase*, convertirla en los tensores correspondientes, inyectarla en el entorno, almacenar la experiencia previa junto con el estado actual, solicitar la acción al agente y enviarla al AGV.

6.3.5. Gestión de las predicciones del modelo

Se ha explicado ya el flujo de mensajes entre *AGVBase* y *RLModel*, pero es necesario hacer una breve mención a la forma en que el AGV procesa la información del modelo. Como se ha mencionado previamente, la clase "*Navigation*" es la que recibe las predicciones por parte del modelo. Se ha desarrollado un mecanismo para no utilizar siempre las acciones, especialmente aquellas que difieren completamente de la trayectoria teórica que debería seguir el AGV. Como se puede observar en el anterior diagrama "Fig 6.16 el objeto acción contiene, además de la acción recibida, la información que determina si realmente ha sido utilizada por el AGV y, en caso contrario, cuál hubiera sido la acción realmente ejecutada.

La manera en que "*Navigation*" decide si una acción es adecuada comienza observando si esta acción ha sido recibida hace más de un tiempo configurable, por defecto establecido en 50 milisegundos. Posteriormente, se analiza cuál debería ser el ángulo de dirección del robot. Si este ángulo supera un umbral configurable, se descarta la acción y se continúa con el sistema de corrección ya implementado. Además de estos dos valores, por motivos de seguridad, si se detecta que la desviación lateral o el ángulo de la máquina es mayor a un cierto umbral, se desactiva temporalmente el uso del modelo. Esto asegura que el AGV no se desvíe significativamente de la trayectoria predefinida, evitando posibles daños graves a la máquina y reduciendo el riesgo de accidentes con personas involucradas.

Con este punto final ya está definido el desarrollo realizado en esta parte práctica del proyecto. Se procederá a ver las mejores pruebas realizadas, algunos de los cambios realizados durante estas.

6.4. Pruebas realizadas

6.4.1. Pruebas con mouse

Como se mencionó al principio, este proyecto tenía como objetivo realizar las pruebas con un AGV similar al de la imagen "Fig 2.1", un robot de tipo *mouse*. Se comenzaron las pruebas con este robot y se observó lo siguiente:

Fig 6.21 Gráfica del valor de perdida *Q*. Recogida por el AGV *mouse*. Fuente propia.

Como se puede observar en la gráfica "Fig 6.21", que muestra el valor de la pérdida *Q*, como eje *y*; en el eje *x* el número de procesos de aprendizaje. Esta gráfica, ya explicada en el punto "6.2.2" debe tender hacia cero, comenzando por un valor alto y disminuyendo a medida que avanza el proceso de aprendizaje. Sin embargo, en esta gráfica se observa que la pérdida no tiende a cero e incluso tiende a aumentar. Por lo tanto, se procede a revisar los valores de recompensa que el entorno devuelve al agente.

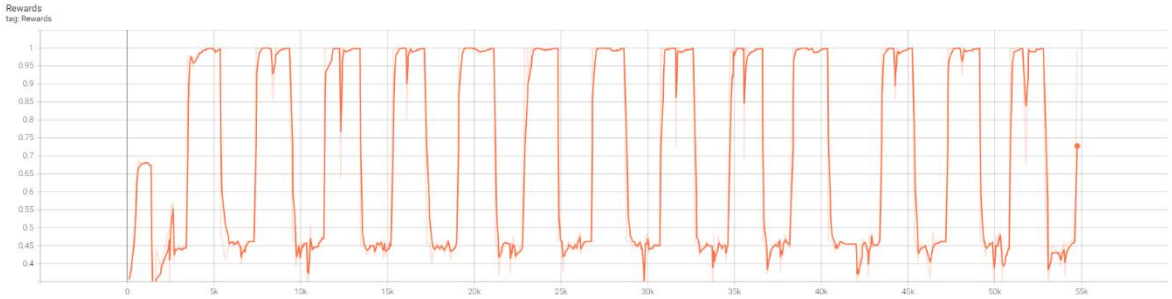

Fig 6.22 Gráfica del valor de recompensa adquirida por el entorno. Recogida por el AGV mouse. Fuente propia.

Como se observa en la gráfica "Fig 6.22" del valor de recompensa devuelto por el entorno se notan fluctuaciones muy anormales. Como se explicó anteriormente, en el punto

“6.3.4.1”, este valor debe oscilar entre cero y uno. Estas fluctuaciones tan bruscas impiden que el proceso de aprendizaje se realice correctamente.

Se analizó en profundidad la razón de estos valores y se descubrió que el componente de localización de la máquina realiza “saltos” de posicionamiento irreales, que no ocurren ni son posibles en la máquina física. Este es el motivo por el cual la recompensa fluctuaba tan bruscamente; cuando la máquina corregía este error irreal causado por la localización, de repente volvía a su posición real.

El comportamiento incorrecto de la posición puede deberse a varios factores: el mapeo realizado puede no ser correcto, la odometría de la máquina puede no estar suficientemente calibrada o incluso podría ser un problema intrínseco del sistema ROS. Aunque, como se comentó en el punto “2.1.1”, ROS es un sistema ampliamente utilizado. Finalmente, se decidió descartar las pruebas físicas con este AGV debido a los retrasos ya sufridos en el desarrollo, y no se pudieron admitir más retrasos por factores ajenos al proyecto.

Como solución, se decidió utilizar otro tipo de AGV, recién fabricado, pero que cuenta con mucha experiencia. A nivel de funcionamiento, en esencia, es muy parecido al tipo *mouse*. Se tratan del AGV de tipo apilador.

6.4.2. Pruebas con apilador

Explicado el motivo del cambio de robot para realizar las pruebas, se procederá a realizar un breve resumen de las diferencias significativas y lo que implica tal acción. El nuevo robot utilizado es una máquina llamada apilador, representada en la imagen “Fig 6.23”. Como se puede observar, visualmente es muy distinta al AGV tipo *mouse*. Este tipo de maquinaria se utiliza generalmente en almacenes de logística con estanterías para colocar contenedores en altura.

Fig 6.23 AGV de tipo apilador. Usado para las pruebas con el modelo RL. Fuente propia.

Con respecto a la parte técnica, el apilador sigue siendo una máquina con movimiento del tipo triciclo, es decir, tiene un motor de dirección, un motor motriz y ruedas fijas que le proporcionan soporte estructural. Por lo tanto, el objetivo principal del proyecto sigue siendo relevante al usar esta máquina. En cuanto al sistema de localización, emplea tecnología de láser guiado por reflectores, ya explicada en el punto “2.1.2.1”.

Una diferencia relevante es el sistema operativo integrado, que en este caso es *Windows*. Esto resulta adecuado para el proyecto, ya que uno de los objetivos planteados era la compatibilidad tanto con este sistema como con *Linux*. En cuanto a la puesta a punto para el funcionamiento del modelo en el robot, los pasos a seguir son muy similares a los mencionados en el punto “6.1.3”, aunque con diferentes comandos. Estos comandos se detallan en los anexos de este proyecto.

Una vez explicado lo que implica el cambio de robot, se procederá a describir las pruebas realizadas y sus resultados. Para comenzar, se ha diseñado un layout muy sencillo que consiste en dos segmentos rectos y una curva. El trayecto que realiza la máquina se abrevia en moverse hacia adelante y hacia atrás siguiendo estos segmentos. Uno de los procesos de aprendizaje ha sido el siguiente:

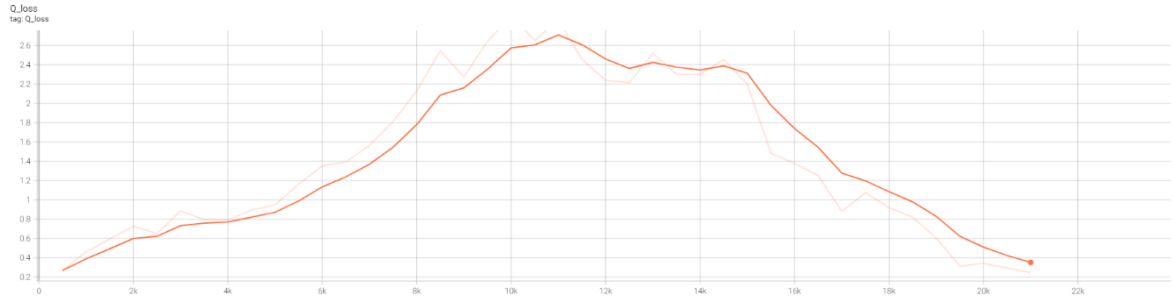

Fig 6.24 Gráfica de la pérdida del valor Q. Recogida por el AGV apilador. Fuente propia.

La gráfica de la "Fig 6.24" muestra una tendencia más normalizada en comparación con las pruebas anteriores. Como se puede observar en el eje *x*, se visualiza el número de procesos de aprendizaje de la máquina, traducido en unidades de tiempo, aproximadamente una hora y media. Este tiempo incluye tanto los movimientos de la máquina como las esperas del proceso de aprendizaje.

Fig 6.25 Gráfica del valor de recompensa adquirida por el entorno. Recogida por el AGV apilador. Fuente propia.

En la gráfica representada en la "Fig 6.25" se observan los valores de recompensa. En esta ocasión, aunque se vea una forma de sierra, en el eje *y* de la gráfica se puede determinar que las diferencias de valores no son muy anómalas. Por lo tanto, se puede concluir que en esta prueba el proceso de aprendizaje se realiza correctamente.

Estos resultados, y los siguientes, se obtienen usando parcialmente el modelo, con un umbral del ángulo admitido bastante elevado. Esto implica que gran parte de las acciones serán ejecutadas por el AGV. Sin embargo, cuando el modelo no puede corregir adecuadamente su desviación lateral, actúa el sistema PID que ya es utilizado por las máquinas. En la "Tabla 6.1" se contemplan las medidas de acciones realizadas, usadas y descartadas.

Número de acciones totales	Acciones usadas	Acciones descartadas	Porcentaje de acciones usadas
15695	13834	1861	88.14

Tabla 6.1 Medidas de las acciones utilizadas por el AGV en la prueba. Fuente propia.

En cuanto a los KPI de la precisión del modelo, se han elaborado gráficas con los datos recogidos. En la "Fig 6.26" se representan las medidas de desviaciones laterales, junto con datos estadísticos, expresados en milímetros, para facilitar la comprensión de los datos almacenados.

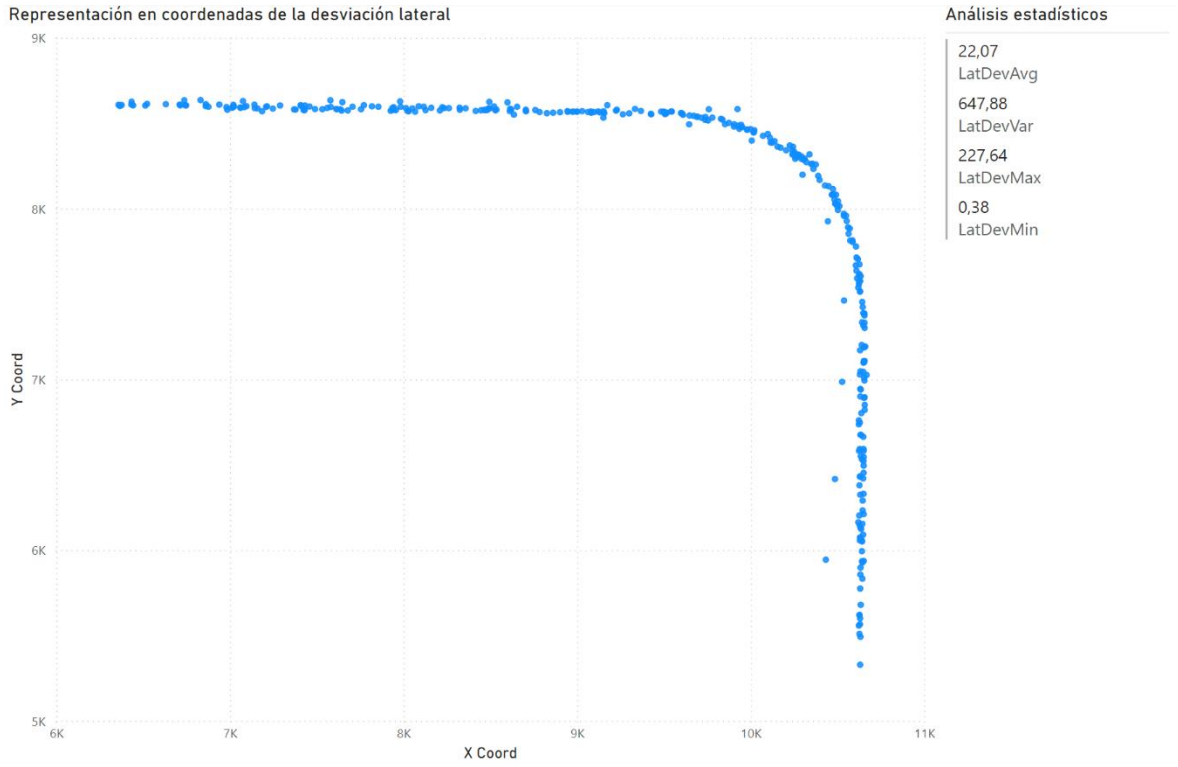

Fig 6.26 Gráfica de desviaciones laterales representados sobre un plano, juntamente con medidas estadísticas. Fuente propia.

Se puede determinar que el modelo, después del aprendizaje, consigue seguir el camino de manera bastante correcta. No obstante, en ciertos puntos se sale del camino (los puntos más distantes de los demás) y no logra retomar la trayectoria por sí mismo. Esto indica cierto "*overfitting*", es decir, el modelo no ha conseguido aprender a generalizar y, al desviarse de la trayectoria preestablecida, no logra reajustarse para volver a esta.

Por este motivo, se muestran las pruebas con el modelo funcionando "parcialmente", ya que, al seguir completamente las acciones del modelo, este no tarda mucho en salirse del trazado y detenerse debido a medidas de seguridad de bajo nivel (los escáneres de seguridad detectan algún obstáculo).

Continuando con la representación de los KPI, se presentan los puntos de parada. En la gráfica "Fig 6.27" se representan los diversos puntos de parada que ha realizado el robot durante las pruebas. Aunque el gráfico es bastante simple, ilustra lo ya comentado sobre la capacidad del modelo para repetir los puntos. Sin embargo, una vez que el robot se pierde por completo, no logra recuperarse.

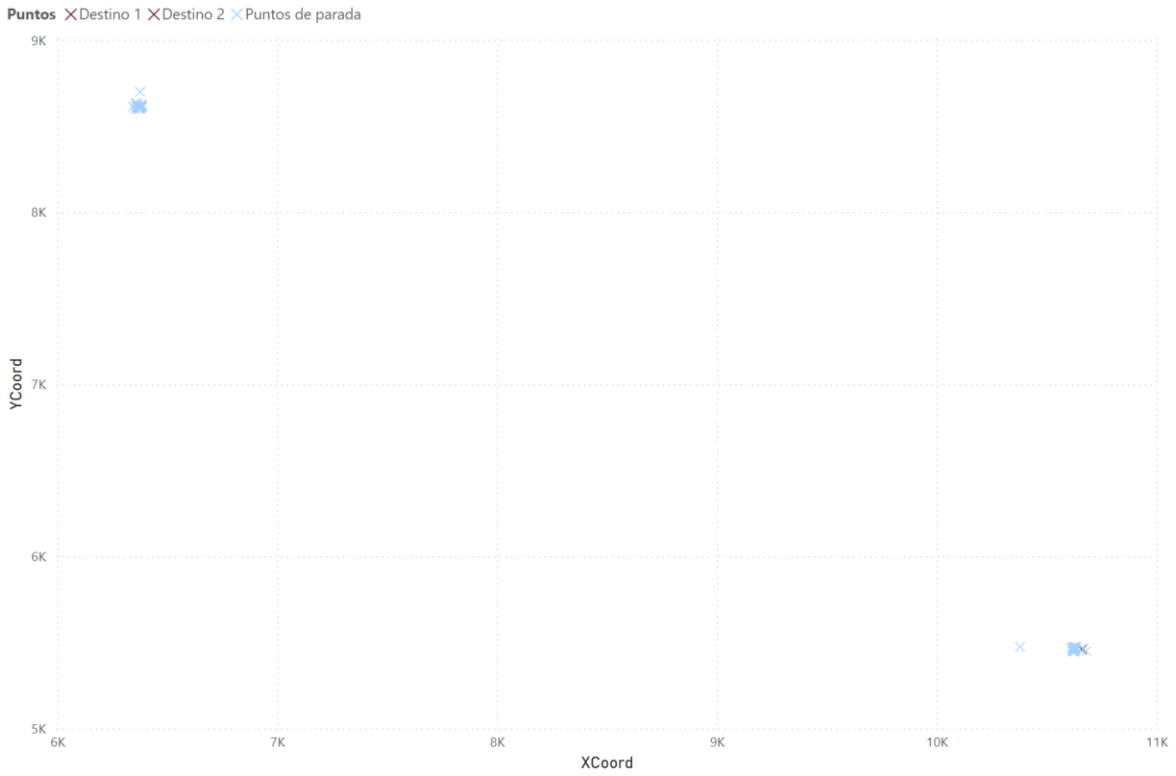

Fig 6.27 Gráfica de puntos de parada representados sobre un plano. Fuente propia.

Antes de continuar con los dos últimos KPI, es importante mencionar el comportamiento visual de la máquina durante el uso de las acciones del modelo. La maximización del valor de recompensa se basa en gran parte en la desviación lateral y la diferencia de ángulo de la máquina, como se representa en la fórmula (6). Durante las pruebas, el AGV se ha comportado de manera muy agresiva en cada paso con respecto al ángulo de dirección, llegando a realizar movimientos bruscos de lado a lado.

Este comportamiento no afecta significativamente el cómputo total en términos de precisión del robot, pero, como es evidente, no es óptimo ni seguro tomar el camino de dicha forma. Por lo tanto, es necesario estudiar más a fondo la técnica de optimización para lograr un equilibrio entre la precisión y la optimización de los movimientos. En los anexos del proyecto se incluirán enlaces a vídeos que muestran el comportamiento descrito y el comportamiento normal de la máquina con el sistema PID.

Continuando con los KPI, se encuentran las medidas de tiempo entre predicciones. En la gráfica “Fig 6.28” se observan distintas medias de diferencia de tiempo entre la petición de predicción y el momento en que realmente llega, como se comentó en el punto “6.2.3”. Además, se incluyen métricas estadísticas relevantes sobre las medidas recogidas, que se pueden extraer una variabilidad del tiempo muy alta. Con estos datos se determina que se obtiene una velocidad muy alta, con una media de poco más de ocho milisegundos e incluso alcanzando mínimos de 10 microsegundos.

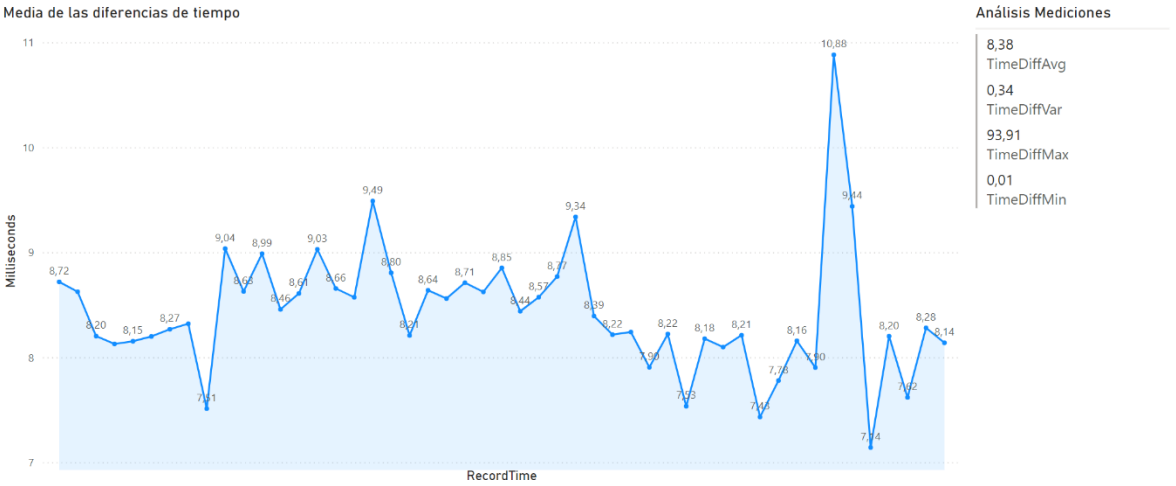

Fig 6.28 Gráfica donde muestra las medias recogidas durante la prueba, junto con las métricas estadísticas. Fuente Propia.

Esta baja latencia se traduce en un uso muy intenso del canal de comunicación, lo que podría acarrear en el colapso de otros dispositivos en comunicación *Ethernet* con el ordenador. Es más relevante obtener medidas de tiempo más regulares que priorizar velocidades más altas y obtener irregularidades, como se observa en la gráfica. Se estudiará el desarrollo de la normalización de estos tiempos para obtener una mayor robustez en este aspecto.

Para finalizar, se analizará el último KPI: el uso de los recursos. Estos están representados en la gráfica de uso de la CPU (“Fig 6.29”). Aspectos relevantes a comentar incluyen la medida utilizada en el eje y, que representa el tanto por 1 de los recursos. Teniendo esto en cuenta, se observan ciertos picos que llegan incluso al 24 por ciento.

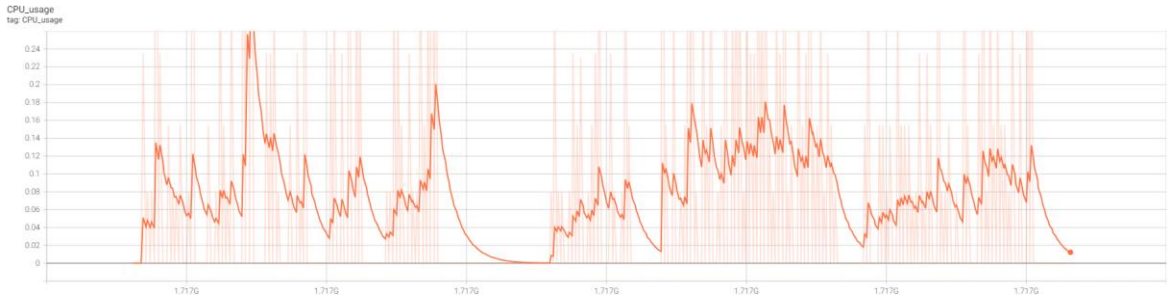

Fig 6.29 Gráfica del uso de la CPU. Fuente propia.

Estos picos son debidos al proceso de aprendizaje, donde el programa requiere de una mayor potencia de computación. A pesar de esto, el uso nominal del modelo se encuentra en un rango del 10-18 por ciento del sistema, siendo una medida admisible según los objetivos propuestos. Aún hay margen de mejora, como se mencionó en el desarrollo del entorno y la estabilización de la demanda de peticiones por parte del AGV.

Por otro lado, continuando con el uso del sistema, se encuentra el uso de memoria del programa. En la siguiente gráfica “Fig 6.30”, se observa el uso habitual de un programa en

relación con la memoria del sistema, utilizando mucha memoria para instanciar todos los recursos y, con el paso del tiempo, optimizando estos hasta regularlos.

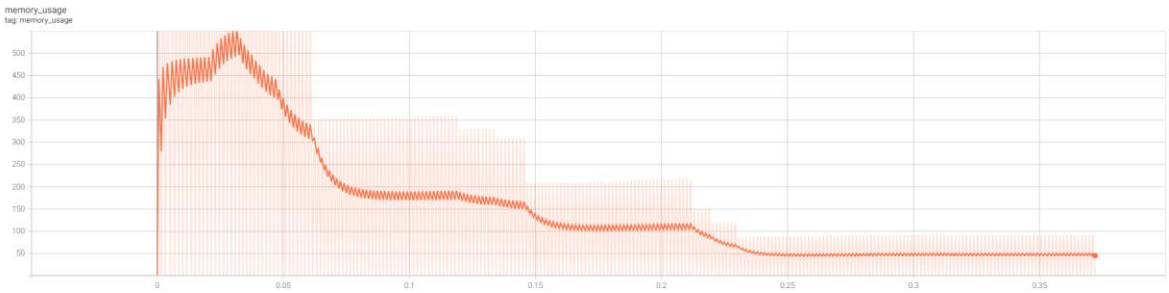

Fig 6.30 Gráfica del uso de la memoria en MB. Fuente propia.

Aquí finaliza la implementación y el ámbito práctico del proyecto. En el transcurso del desarrollo, se han mencionado ciertas mejoras y conclusiones de cara al futuro que se verán profundizadas en puntos posteriores.

7. Conclusiones

En esta sección se desarrollarán las deducciones del trabajo realizado en este proyecto, repasando tanto la parte teórica, que incluye la investigación de la solución propuesta, como la parte práctica, que abarca el desarrollo, implementación y pruebas del modelo de aprendizaje por refuerzo.

Al ser un proyecto enfocado principalmente en la investigación, se ha llevado a cabo un trabajo de investigación bastante exhaustivo del amplio campo de la inteligencia artificial. Se han revisado distintas referencias para lograr comprender y entender este vasto mundo que se encuentra en constante desarrollo desde hace varios años y gracias a la mejora de la tecnología y la potencia computacional, este campo está experimentando un auge sin precedentes.

Uno de los pilares más investigados ha sido el aprendizaje por refuerzo, desde sus bases hasta la investigación de diversos algoritmos para encontrar la manera óptima de alcanzar los objetivos. Se concluye que la investigación realizada es bastante plausible, gracias también a las diversas referencias consultadas. Sin embargo, esto no es suficiente para detener el continuo aprendizaje que este campo ofrece.

Respecto al desarrollo, se han completado varios objetivos que habían sido marcado desde un principio, aunque otros no se lograron debido a la falta de experiencia en la materia, razón que causó varios retrasos, especialmente en el planteamiento de la solución, la comprensión de los algoritmos y las implementaciones investigadas. Debido a ello, se dejó de lado la parte más visual del proyecto y las pruebas en entornos reales de la industria, dado que el objetivo principal del proyecto tenía más peso y actualmente se encuentra parcialmente completado.

No obstante, extrayendo los KPI planteados desde un inicio, se ha conseguido obtener mucha información y comprender el funcionamiento interno de lo desarrollado, con miras a mejorar la eficiencia del sistema.

Este proyecto ha sido todo un reto y una primera toma de contacto con el sector del aprendizaje por refuerzo, utilizando herramientas bastante complejas y una comprensión amplia del sistema. Aun así, se ha logrado un despliegue significativo del desarrollo del modelo, incluida la totalidad de la comunicación y la gestión de las predicciones del agente. Aunque los resultados no se ajusten a el nivel de ejecución estimado, este último logro otorga un valor considerable al trabajo final realizado.

Desde la perspectiva empresarial, Artisteril no apresura en que la navegación con un modelo de inteligencia artificial se desarrolle completamente en este proyecto. Sin embargo, se tiene la certeza de que, con más recursos y tiempo dedicado exclusivamente al desarrollo, se conseguirá una solución bastante plausible, resultando en una mejora y una técnica publicitaria importante. Por ello, este proyecto pronto seguirá adelante y se realizarán más avances.

A mi parecer, este trabajo ha sido todo un reto y, en ocasiones, bastante complejo, puesto que no se ha elaborado en el transcurso de la carrera temarios ajustados a los explorados en este trabajo. Sin embargo, mi dedicación y estudio dedicado son demostrables. Además de un interés que será continuado en la experimentación de técnicas y aprendizajes personales en relación a este fascinante campo.

8. Mejoras y ampliaciones

En esta sección se tratarán diversas opciones de mejoras y ampliaciones a realizar en el trabajo. Se obviarán los objetivos pendientes que ya formaban parte de la planificación inicial, dado que estos son requisitos esenciales para el pleno funcionamiento del proyecto.

8.1. Mejoras en el desarrollo

Haciendo mención del punto “6.3.4.1”, la mejora del entorno es imperativa y no solo en términos de optimización del traspaso de información. Se puede considerar la migración del componente de entorno al programa base del AGV, ya que este es el que tiene información completa del estado del AGV y del entorno en el que se encuentra. La responsabilidad del entorno en el programa del modelo RL es el cálculo del valor de recompensa, aunque en realidad este valor se puede calcular directamente en el programa en *C#*, reduciendo así el doble procesamiento de una información que ya se encuentra en la base del sistema.

Este planteamiento podría acarrear una mejora sustancial en el uso de los recursos y reducir las responsabilidades del lenguaje *Python*, que, al ser interpretado, consume más recursos que *C#*, que es un lenguaje compilado. Además, evitaría la duplicación de lógica en diferentes puntos independientes, lo que supone una mejora a nivel de diseño y mantenibilidad del código.

Se ha planteado otro cambio de paradigma con respecto al desarrollo actual, que consiste en el planteamiento continuo del espacio de acciones. Como se ha explicado en el punto “2.4.2 actualmente el modelo debe tomar una decisión entre una infinidad de acciones posibles. Sin embargo, ¿qué pasaría si se cambiara el paradigma de acciones infinitas a un conjunto finito de, por ejemplo, tres posibles acciones? Estas acciones pueden ser: mover dirección derecha, mover dirección izquierda y no moverse.

Cada movimiento sería una unidad fija, como por ejemplo medio grado. Dado que se trata de una transmisión de datos de alta velocidad, no debería haber problemas de falta de movimiento e incluso podría hacer que el comportamiento del robot sea más fluido en comparación con el actual. Esto podría conllevar el uso de algoritmos más simples y sencillos, pero no por ello menos potentes.

8.2. Mejoras en las redes neuronales

Como ya se mencionó en el punto “6.3.4.3”, las redes neuronales actuales son muy sencillas y simples, lo que puede contribuir al problema de “*overfitting*” observado en las pruebas realizadas. A pesar de esto, sabiendo la causa del problema, es necesario profundizar mucho

más en el diseño de redes neuronales. Hasta ahora, se ha investigado superficialmente lo que cada una realiza y cómo se usa, pero el conjunto de varias redes se escapa del entendimiento actual.

Por lo tanto, primero se debe realizar el trabajo de mejorar las bases del proyecto, incluyendo el propio diseño y la implementación del modelo de aprendizaje por refuerzo, mencionado anteriormente. De esta manera, se podrá avanzar en mejoras más complejas. Aun así, como ya se ha comentado, se ha estado investigando las redes neuronales recurrentes, en especial las LSTM, que pueden proporcionar una mejora adicional al modelo.

8.3. Mejoras en el proceso de aprendizaje

El proceso de aprendizaje actual es lento y tedioso. Como se ha observado en las pruebas, una única prueba donde el aprendizaje se ha realizado correctamente ha tardado más de una hora y media en conseguir buenos resultados. Esto sin mencionar que solo se ha tratado del movimiento en una curva. Este enfoque no es óptimo ni rentable en ningún sentido. Por estos motivos, se pondrá especial énfasis en la implementación de la herramienta de simulación del AGV.

La simulación es una herramienta ya creada actualmente, pero está en fase de mejoras, especialmente en la virtualización de dispositivos enteramente físicos, como los *encoder* de los diversos motores. Además de reproducir virtualmente el robot para que se asemeje a la realidad, también es interesante para simular posibles errores o malfuncionamientos de la máquina. De esta manera, se puede realizar el proceso de aprendizaje de manera segura, sin dañar ningún dispositivo real.

Una herramienta de simulación altamente desarrollada puede hacer que el aprendizaje del modelo sea más sencillo y seguro. Además, permite escalar los procesos, pudiendo realizar entrenamientos de redes neuronales muy extensos. Al correr en un entorno de simulación, es posible ajustar los pesos y sesgos en la mayoría de las capas de la red. Una vez que la red se transfiere a la máquina real, solo se necesitaría entrenar unas capas finales para realizar lo que se denomina el ajuste fino del modelo. Esto mejora el rendimiento, al realizar el proceso de entrenamiento en partes de redes neuronales sencillas, sin perder la complejidad de la red pre-entrenada.

8.4. Mejoras en el hardware

Como ampliación de futuro, se considera la posibilidad de añadir hardware más potente y óptimo para el uso de redes neuronales. Como se explicó en el punto “6.3.1”, la base de las comunicaciones utiliza *Ethernet*, lo que permite una gran escalabilidad y la posibilidad de utilizar dispositivos externos para el procesamiento aislado del modelo. Actualmente, no se está aprovechando la plataforma *CUDA* que ofrece *Pytorch*, debido a la falta de una tarjeta

gráfica dedicada en el ordenador del AGV, ya que solo se dispone de la integrada en la CPU, que no soporta esta plataforma.

Esta es una de las mejoras para las que ya se ha realizado una pequeña investigación. Un ejemplo de esto son los dispositivos "*Nvidia Jetson Nano*", especializados en el desarrollo de diversos proyectos relacionados con el *deep learning*.

9. Bibliografía

[1] A. J. Moshayedi, J. Li, y L. Liao, «AGV (automated guided vehicle) robot: Mission and obstacles in design and performance», *J. Simul.*, 2019.

[2] T. J. Chong, X. J. Tang, C. H. Leng, M. Yogeswaran, O. E. Ng, y Y. Z. Chong, «Sensor Technologies and Simultaneous Localization and Mapping (SLAM)», *Procedia Comput. Sci.*, vol. 76, pp. 174-179, 2015, doi: 10.1016/j.procs.2015.12.336.

[3] «Model predictive speed and steering control — PythonRobotics documentation». Accedido: 9 de marzo de 2024. [En línea]. Disponible en: https://atsushisakai.github.io/PythonRobotics/modules/path_tracking/model_predictive_speed_and_steering_control/model_predictive_speed_and_steering_control.html

[4] «Controlador PID - Control Automático - Picuino». Accedido: 10 de marzo de 2024. [En línea]. Disponible en: https://www.picuino.com/es/control-pid.html

[5] A. Turing, «Computing Machinery and Intelligence», 1950. Accedido: 13 de diciembre de 2023. [En línea]. Disponible en: https://redirect.cs.umbc.edu/courses/471/papers/turing.pdf

[6] J. McCarthy, «WHAT IS ARTIFICIAL INTELLIGENCE?», 2007, Accedido: 11 de diciembre de 2023. [En línea]. Disponible en: https://www-formal.stanford.edu/jmc/whatisai.pdf

[7] S. Russell y P. Norvig, *Artificial Intelligence: A Modern Approach*, *4th US ed.* en Pearson series in artificial intelligence. University of California, Berkeley, 2021.

[8] G. W. Ng y W. C. Leung, «Strong Artificial Intelligence and Consciousness», *J. Artif. Intell. Conscious.*, vol. 07, n.º 01, pp. 63-72, mar. 2020, doi: 10.1142/S2705078520300042.

[9] «What is Strong AI? | IBM». Accedido: 16 de diciembre de 2023. [En línea]. Disponible en: https://www.ibm.com/topics/strong-ai

[10] «What is Machine Learning? | IBM». Accedido: 16 de diciembre de 2023. [En línea]. Disponible en: https://www.ibm.com/topics/machine-learning

[11] E. Alpaydin, *Introduction to machine learning*, 2nd ed. en Adaptive computation and machine learning. Cambridge, Mass: MIT Press, 2010.

[12] Z. Ghahramani, «Unsupervised Learning», en *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*, O. Bousquet, U. von Luxburg, y G. Rätsch, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 72-112. doi: 10.1007/978-3-540-28650-9_5.

[13] «What is Unsupervised Learning? | IBM». Accedido: 10 de marzo de 2024. [En línea]. Disponible en: https://www.ibm.com/topics/unsupervised-learning

[14] R. Verma, V. Nagar, y S. Mahapatra, «Introduction to Supervised Learning», en *Data Analytics in Bioinformatics*, 2021, pp. 1-34. doi: 10.1002/9781119785620.ch1.

[15] «What is Supervised Learning? | IBM». Accedido: 19 de diciembre de 2023. [En línea]. Disponible en: https://www.ibm.com/topics/supervised-learning

[16] «What is Deep Learning? | IBM». Accedido: 20 de diciembre de 2023. [En línea]. Disponible en: https://www.ibm.com/topics/deep-learning

[17] S. Razavi, «Deep learning, explained: Fundamentals, explainability, and bridgeability to process-based modelling», *Environ. Model. Softw.*, vol. 144, p. 105159, oct. 2021, doi: 10.1016/j.envsoft.2021.105159.

[18] Y. LeCun, Y. Bengio, y G. Hinton, «Deep learning», *Nature*, vol. 521, n.º 7553, pp. 436-444, may 2015, doi: 10.1038/nature14539.

[19] «Step by step VGG16 implementation in Keras for beginners | by Rohit Thakur | Towards Data Science». Accedido: 10 de marzo de 2024. [En línea]. Disponible en: https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c

[20] «Train a software agent to behave rationally with reinforcement learning - IBM Developer». Accedido: 1 de enero de 2024. [En línea]. Disponible en: https://developer.ibm.com/articles/cc-reinforcement-learning-train-software-agent/

[21] R. S. Sutton y A. G. Barto, *Reinforcement learning: an introduction*. en Adaptive computation and machine learning. Cambridge, Mass: MIT Press, 1998.

[22] T. Hester, M. Quinlan, y P. Stone, «RTMBA: A Real-Time Model-Based Reinforcement Learning Architecture for robot control», en *2012 IEEE International Conference on Robotics and Automation*, St Paul, MN, USA: IEEE, may 2012, pp. 85-90. doi: 10.1109/ICRA.2012.6225072.

[23] M. van Otterlo y M. Wiering, «Reinforcement Learning and Markov Decision Processes», en *Reinforcement Learning: State-of-the-Art*, M. Wiering y M. van Otterlo, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 3-42. doi: 10.1007/978-3-642-27645-3_1.

[24] S. R. Eddy, «What is dynamic programming?», *Nat. Biotechnol.*, vol. 22, n.º 7, pp. 909-910, jul. 2004, doi: 10.1038/nbt0704-909.

[25] B. O'Donoghue, I. Osband, R. Munos, y V. Mnih, «The Uncertainty Bellman Equation and Exploration».

[26] A. M. Johansen, L. Evers, y N. Whiteley, «Monte carlo methods», *Int. Encycl. Educ.*, pp. 296-303, 2010.

[27] «Models for machine learning - IBM Developer». Accedido: 3 de enero de 2024. [En línea]. Disponible en: https://developer.ibm.com/articles/cc-models-machine-learning/#reinforcement-learning

[28] J. Fan, Z. Wang, Y. Xie, y Z. Yang, «A Theoretical Analysis of Deep Q-Learning». arXiv, 23 de febrero de 2020. Accedido: 3 de enero de 2024. [En línea]. Disponible en: http://arxiv.org/abs/1901.00137

[29] B. Liu, Q. Cai, Z. Yang, y Z. Wang, «Neural Trust Region/Proximal Policy Optimization Attains Globally Optimal Policy», en *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, y R. Garnett, Eds., Curran Associates, Inc., 2019. [En línea]. Disponible en: https://proceedings.neurips.cc/paper_files/paper/2019/file/227e072d131ba77451d8f27ab9afdfb7-Paper.pdf

[30] «Practical Guide to Clustering Algorithms & Evaluation in R Tutorials & Notes | Machine Learning | HackerEarth». Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/clustering-algorithms-evaluation-r/tutorial/

[31] M. M. Wolf, «Mathematical foundations of supervised learning», 2023.

[32] B. Pang, E. Nijkamp, y Y. N. Wu, «Deep Learning With TensorFlow: A Review», *J. Educ. Behav. Stat.*, vol. 45, n.º 2, pp. 227-248, abr. 2020, doi: 10.3102/1076998619872761.

[33] «Train a Deep Q Network with TF-Agents | TensorFlow Agents». Accedido: 9 de enero de 2024. [En línea]. Disponible en: https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial

[34] «PyTorch 2.0 | PyTorch». Accedido: 10 de enero de 2024. [En línea]. Disponible en: https://pytorch.org/get-started/pytorch-2.0/#distributed

[35] N. Ketkar, *Deep Learning with Python*. Berkeley, CA: Apress, 2017. doi: 10.1007/978-1-4842-2766-4.

[36] «Keras: Deep Learning for humans». Accedido: 10 de enero de 2024. [En línea]. Disponible en: https://keras.io/keras_3/

[37] G. Brockman *et al.*, «OpenAI Gym». arXiv, 5 de junio de 2016. Accedido: 9 de enero de 2024. [En línea]. Disponible en: http://arxiv.org/abs/1606.01540

[38] Z. Zhu *et al.*, «Pearl: A Production-ready Reinforcement Learning Agent». arXiv, 6 de diciembre de 2023. Accedido: 12 de enero de 2024. [En línea]. Disponible en: http://arxiv.org/abs/2312.03814

[39] «Reinforcement Learning (DQN) Tutorial — PyTorch Tutorials 2.3.0+cu121 documentation». Accedido: 31 de mayo de 2024. [En línea]. Disponible en: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

[40] «Reinforcement Learning (PPO) with TorchRL Tutorial — PyTorch Tutorials 2.3.0+cu121 documentation». Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://pytorch.org/tutorials/intermediate/reinforcement_ppo.html

[41] «Pendulum: Writing your environment and transforms with TorchRL — PyTorch Tutorials 2.3.0+cu121 documentation». Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://pytorch.org/tutorials/advanced/pendulum.html

[42] S. Gu, T. Lillicrap, I. Sutskever, y S. Levine, «Continuous Deep Q-Learning with Model-based Acceleration». arXiv, 2 de marzo de 2016. Accedido: 28 de abril de 2024. [En línea]. Disponible en: http://arxiv.org/abs/1603.00748

[43] BY571, «BY571/Normalized-Advantage-Function-NAF-». 27 de abril de 2024. Accedido: 28 de abril de 2024. [En línea]. Disponible en: https://github.com/BY571/Normalized-Advantage-Function-NAF-

[44] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more*. Packt Publishing Ltd, 2020.

[45] «PacktPublishing/Deep-Reinforcement-Learning-Hands-On-Second-Edition: Deep-Reinforcement-Learning-Hands-On-Second-Edition, published by Packt». Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://github.com/PacktPublishing/Deep-Reinforcement-Learning-Hands-On-Second-Edition/tree/master

[46] J. Highsmith, «History: The Agile Manifesto». Accedido: 12 de enero de 2024. [En línea]. Disponible en: https://agilemanifesto.org/history.html

[47] «Download .NET 8.0 (Linux, macOS, and Windows)». Accedido: 15 de marzo de 2024. [En línea]. Disponible en: https://dotnet.microsoft.com/en-us/download/dotnet/8.0

[48] «LiteDB :: A .NET embedded NoSQL database». Accedido: 29 de mayo de 2024. [En línea]. Disponible en: http://www.litedb.org

[49] T. Hombergs, «Hexagonal Architecture with Java and Spring». Accedido: 31 de mayo de 2024. [En línea]. Disponible en: https://reflectoring.io/spring-hexagonal/

[50] «Netron». Accedido: 31 de mayo de 2024. [En línea]. Disponible en: https://netron.app/