Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

# Trabajo de Fin de Grado

Grado en Ingeniería Informática

## Creación de entornos para desplazamiento inteligente de robot con uso de aprendizaje automático

*Creation of environments for intelligent robot movement using machine learning techniques*

Manuel Andrés Carrera Galafate

La Laguna, *8* de *septiembre* de 2021

D. **Toledo Carrillo, Jonay Tomas**, con N.I.F. 78.698.554-Y profesor Titular de Universidad adscrito al Departamento de Ingeniería informática y sistemas de la Universidad de La Laguna, como tutor

**C E R T I F I C A**

Que la presente memoria titulada:

*"Creación de entornos para desplazamiento inteligente de robot con uso de aprendizaje automático"*

ha sido realizada bajo su dirección por D. **Manuel Andrés Carrera Galafate**,

con N.I.F. 45.853.589-F.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 21 de noviembre de 2021

# Agradecimientos

Agradezco a todas las personas presentes durante esta etapa de investigación para la finalización de mi Trabajo de Fin de Grado y también del grado en sí, hayan estado involucrado de manera más o menos directa.

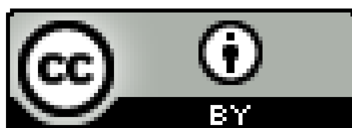
En primer lugar a todos los amigos que he hecho en la carrera que me han hecho el camino más ameno y disfrutable y que junto al profesorado me han hecho aprender mucho y me han hecho crecer académica y vitalmente.

A Jonay, por ser un gran tutor, por su ayuda en la planificación, organización y desarrollo de este Trabajo de fin de Grado.

Y por supuesto mis padres por apoyarme en todos los sentidos a pesar de las dificultades, y más en los momentos que estamos atravesando de pandemia mundial, junto a mi hermano Rafael por ser la motivación principal a lo largo de este periodo para seguir adelante en los momentos más difíciles del camino.

A todos, muchas gracias por estar ahí y verme cerrar esta etapa de mi vida.

# Licencia




© Esta obra está bajo una licencia de Creative Commons

Reconocimiento 4.0 Internacional.

**Resumen**

Dentro de los subsistemas principales de un robot móvil, se encuentra el sistema de navegación. Este sistema es el encargado de definir una ruta desde el punto actual donde se encuentra el robot, hasta su destino, esquivando obstáculos y evitando situaciones de riesgo.

Este TFG se centrará en el estudio de la creación de un planificador dinámico para robots móviles utilizando herramientas de aprendizaje automático, o machine learning. Se utilizará un escenario de simulación de robots móviles en entornos complejos, para entrenar el mejor comportamiento posible a partir de una serie de simulaciones con comportamiento aleatorio. De esta manera se podrá usar este planificador dinámico como base para generar uno en el futuro capaz de esquivar obstáculos, no basado únicamente en la posición actual de estos obstáculos, sino en la posición que tendrán en un futuro próximo, evitando posibles colisiones con antelación.

El carácter de esta memoria es mayormente práctico, nuestro fin es documentar los pasos que se han seguido para componer el sistema para que estos sean reproducibles en el futuro, logrando así poder entrenar otros sistemas robóticos, entrenar mediante el uso de otros algoritmos o arquitecturas de red neuronal, y en general inspirar al lector a realizar su propia experimentación con sistemas compuestos por robótica e inteligencia artificial.

**Palabras clave:** Robótica, Aprendizaje reforzado, Inteligencia artificial, Algoritmia.

**Abstract**

In the main subsystems of a mobile robot, there is the navigation system. This system is responsible for defining a route from the current location of the robot to the destination, avoiding obstacles and preventing risky situations.

This TFG will focus on the study of the creation of a dynamic planner for mobile robots using machine learning tools. A simulation scenario of mobile robots in complex environments will be used to train the best possible outcome from a series of simulations with random behavior. In this way, it will be possible to use this dynamic planner as a basis to create in the future a one capable of avoiding obstacles, not only based on the current position of these obstacles, but also on the position they will have in the near future, avoiding possible collisions in advance.

The character of this report is mostly practical, our goal is to document the steps that have been followed to compose the system so that these are reproducible in the future, thus making it possible to train other robotic systems, train using other algorithms or neural network architectures, and in general give some inspiration to the reader to perform their own experimentation with systems composed of robotics and artificial intelligence.

**Keywords:** Robotics, Reinforcement learning, Artificial intelligence, Algorithms.

# Índice general


**1. Introducción** **11**

1. 1. Motivación 11

1. 2. Antecedentes 11

1. 3. Problemática. 12

**2. Objetivo** **13**

2. 1. Objetivos generales 13

2. 2. Objetivos específicos 13

**3. Robot y modelización.** **15**

**4. Preparación y desarrollo del proyecto.** **17**

4. 1. Familiarización y preparación del entorno. 17

4. 2. Lista de comandos ROS frecuentemente utilizados. 19

4. 3. Directorio de trabajo Catkin. 19

4. 4. Creación de paquete catkin 20

4. 5. Primera ejecución de Gazebo 22

4. 6. Tópicos ROS. 23

**5. Aprendizaje automático y tratamiento de datos.** **26**

5. 1. Q-Learn 28

**6. Desarrollo python.** **29**

6. 1. Gym AI. 29

6. 2. Estructura de clases de los robots. 29

**7. Problemáticas y posibles soluciones.** **34**

7. 1. Problemas de la simulación. 34

7. 2. Problemas de diseño del aprendizaje reforzado. 35

7. 3. Problemas de Software. 36

**8. Experimentación y resultados.** **37**

**9. Conclusiones y líneas futuras de trabajo.** **43**

**9. Summary and conclusions.** **45**

**10. Presupuesto** **47**

**11. Bibliografía** **49**

# Índice de figuras


Figura 2.1: Imágenes del Pioneer3dx 13

Figura 2.2: Visualización del área del sensor láser (vista cenital y vista general). 14

Figura 3.1: Medidas del robot. 15

Figura 3.2: Vista cenital que representa el sensor del robot. 16

Figura 4.1: Distribuciones ROS 18

Figura 4.2: Contenido de package.xml. 21

Figura 4.3: Código de ejemplo de un launchfile que contenga un mundo vacío y nuestro robot. 22

Figura 4.4: Gazebo con robot y mundo vacío 23

Figura 4.5: Lista de tópicos disponibles 24

Figura 4.6: Ejemplo de suscripción y publicación de tópicos. 25

Figura 5.1: Diagrama representativo del vector de ejemplo. 27

Figura 6.1: Diagrama UML de las clases principales involucradas para representar el robot. 31

Figura 7.1: Representación ganancia (eje vertical) para para el algoritmo Q-Learn cada 20 iteraciones (eje horizontal). 35

Figura 8.1: Mapa de entrenamiento. 37

Figura 8.2: Esquema de aprendizaje 38

Figura 8.3: Simulador y gráfica de aprendizaje para 10.000 episodios de entrenamiento. 40

Figura 8.4: Gráfica de aprendizaje del algoritmo Q-Learn de Acutronic Robotics. 41

# Índice de tablas

Tabla 4.1: Lista de comandos 18

Tabla 4.2: Lista de comandos 19

Tabla 10.1: Desglose de horas de trabajo 47

Tabla 10.2: Desglose de precios del Hardware utilizado. 48

# 1. Introducción

## 1. 1. Motivación

"¿Heredarán los robots la Tierra? Sí, pero serán nuestros hijos."

Marvin Minsky (1994).

La robótica es a partes iguales fascinante y compleja. Es una rama de distintos campos de la ciencia, que combina muchas de estas áreas como pueden ser la mecánica, la electrónica, la informática, la inteligencia artificial, la ingeniería de control, las matemáticas o la física para dar soluciones a problemas de todo tipo. Pueden ser tareas sencillas para un ser humano de a pie, como colocar piezas de un coche en una fábrica, pero que acarrean ciertas problemáticas, como la repetición innecesaria, o pueden resolver problemas más complejos, como podría ser resolver un cubo de rubik. La robótica es una de las máximas expresiones de la ingeniería, y como tal uno de los temas más adecuados para cerrar este grado.

## 1. 2. Antecedentes

La **robótica** como rama científica al igual que la informática es relativamente reciente y sin quedarse atrás también ha experimentado grandes avances en un periodo muy corto de tiempo. Aprendimos en la asignatura en robótica computacional la existencia de algoritmos que nos permiten resolver problemas de movilidad aparentemente triviales que requieren la combinación de muchos datos en forma de distintas estructuras de datos, pero siempre a nivel teórico, obviando las posibles molestias e inconvenientes del medio físico, y suponiendo que nunca tengamos que medir posibles cambios en el mundo. También hemos visto a lo largo del grado las distintas definiciones de **inteligencia artificial**, su aplicabilidad, sus distintos tipos de implementaciones, y más concretamente, para lo que nos atañe en este trabajo de final de grado, el uso y efectividad del

**aprendizaje automático** en ciertos contextos.

"Un robot es una **máquina controlada por ordenador y programada para moverse, manipular objetos y realizar trabajos** a la vez que interacciona con su entorno. El robot a veces **recuerda a los seres humanos** y es capaz de efectuar diversas tareas humanas complejas cuando se les indica que lo hagan, o por habérselas programado con antelación." [1]

Parece que sin importar dónde busquemos una definición de robótica, esta siempre tendrá asociado de una manera implícita o explícita un componente de inteligencia, y es por ello que no es extraño que esté tan extendida la combinación de estas dos ramas de la informática. De hecho, ahí donde veamos noticias relacionadas con avances en robótica, siempre veremos avances en inteligencia artificial.

## 1. 3. Problemática.

La robótica móvil es de gran interés en distintos ámbitos, como pueden ser la producción industrial, aplicación doméstica o incluso más recientemente la automoción como en el caso de los coches autónomos de Tesla.

Dado que el fin de estos sistemas es el de dar respuestas automatizadas a problemas, con el fin de conseguir sistemas más flexibles, capaces de funcionar en distintos ambientes, necesitamos o bien algoritmos que puedan tener en cuenta todas las casuísticas del entorno real, que puede llegar a ser muy complejo dependiendo del tipo de tarea a realizar y de cómo se abstraiga el problema pero que con una buena codificación nos darán comportamientos previsibles y robustos, o intentar hacer uso de tecnologías que nos permitan dotar de cierta inteligencia a nuestras máquinas, con menos previsibilidad, pero que pueden llegar a darnos una gran flexibilidad y escalabilidad para un gran número de problemas. En este Trabajo de Fin de Grado se optará por implementar la segunda opción, basado en la tendencia de los sistemas actuales de robótica, que se orientan a sistemas cada vez más inteligentes, se verá la viabilidad de implementación de estos en sistemas que son resolubles también por algoritmos clásicos, e intentaremos hacer un análisis sobre la adecuación de cada uno de estos para el problema que queramos resolver.

# 2. Objetivo

## 2. 1. Objetivos generales

El fin de este proyecto es el de preparar un entorno que nos permita ver el comportamiento simulado de un robot que tenga instalado un sistema dinámico de navegación, experimentar y probar con él en busca de obtener los mejores resultados posibles. Luego analizaremos estos y comprobaremos la utilidad y posibilidades que nos ofrecen este tipo de sistemas para robots móviles.

## 2. 2. Objetivos específicos

Para preparar el entorno se hará uso de las librerías de **ROS** [2] (Robots Operating System) que son una serie de librerías orientadas a desarrollar robótica y **Gazebo** [3] que es un simulador de robótica 3D, incluyendo simulación física. Ambas librerías nos permitirán trabajar de una manera más sencilla y cómoda sin tener que llegar a realizar desarrollos a bajo nivel con nuestro robot, que concretamente será el **Pioneer3dx**. Las librerías se han escogido por cuestiones de popularidad, y el modelo del robot es uno del que la Universidad de La Laguna dispone, pudiendo ponerlo a nuestra disposición para hacer las pruebas en el entorno real para la fase final de este proyecto si fuera necesario.

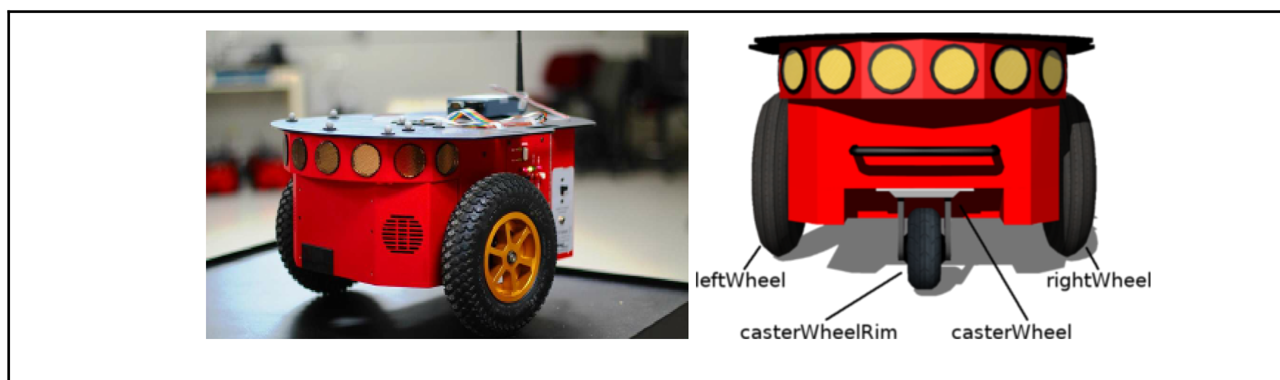



Figura 2.1: Imágenes del Pioneer3dx

El sistema dinámico que se plantea hará uso de técnicas de **aprendizaje reforzado.** Nuestros robots simulado y real disponen de un sensor láser cada uno que es capaz de detectar una serie de puntos en un plano a una determinada distancia. Esta información es retornada como un vector de valores que indican la distancia a la que se encuentran objetos que choquen con la trayectoria del láser. Es por esto que consideramos que ya que en el proyecto se va a trabajar con una cantidad considerable de datos, el uso del aprendizaje reforzado es plausible y pueda llegar a hacernos obtener mejores resultados que usando algoritmos clásicos.

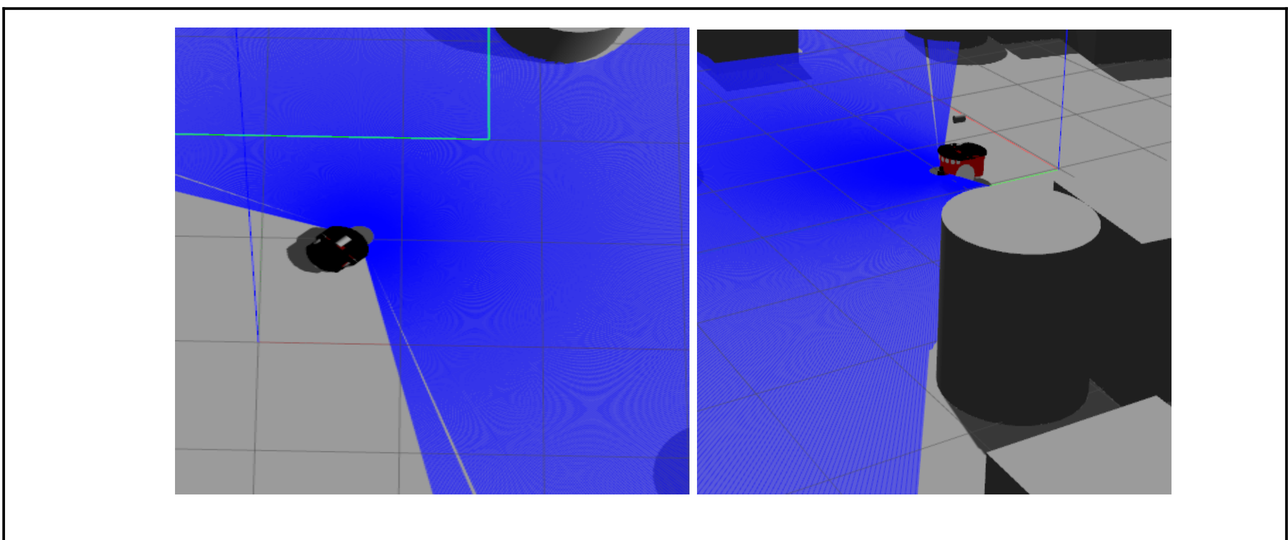


Figura 2.2: Visualización del área del sensor láser (vista cenital y vista general).

La memoria de este trabajo de fin de grado es una recopilación de todos los pasos realizados para lograr el objetivo y que estos sean reproducibles por el lector; se alternará mucho entre explicaciones y fragmentos de código, además de que se intentará seguir una cronología lo más lógica y acorde posible a cómo se realizó realmente, aunque puede que no sea totalmente fiel en algunos puntos ya que a posteriori se ha valorado que hay maneras más eficientes o racionales de realizar estas acciones.

Se debe tener en cuenta que esto es un proyecto enfocado al desarrollo en alto nivel de mejora del planificador, no nos fijaremos en aspectos como en cómo se mueve el robot, cálculos físicos de las colisiones y movimientos, funcionamiento del láser, etc., ya que se entiende que no corresponde en la extensión de un TFG.

# 3. Robot y modelización.

Como hemos adelantado en la introducción, el robot con el que trabajaremos es el **Pioneer3dx.**

"El Pioneer 3-DX es un pequeño y ligero robot de dos ruedas y dos motores con accionamiento diferencial de dos ruedas y dos motores, ideal para su uso en laboratorios o aulas. El robot viene completo con SONAR frontal, una batería, codificadores de rueda, un microcontrolador con firmware ARCOS y el paquete de desarrollo de software de robótica móvil avanzado Pioneer SDK, paquete de desarrollo de software de robótica móvil de Pioneer.

Los robots de investigación Pioneer son los robots móviles inteligentes más populares del mundo para la educación y la investigación. Su versatilidad, fiabilidad y fiabilidad y durabilidad los han convertido en la plataforma preferida para la inteligencia artificial. Los Pioneer son premontados, personalizables, actualizables y/o suficientemente resistentes como para durar años de uso en laboratorios y aulas." [4]

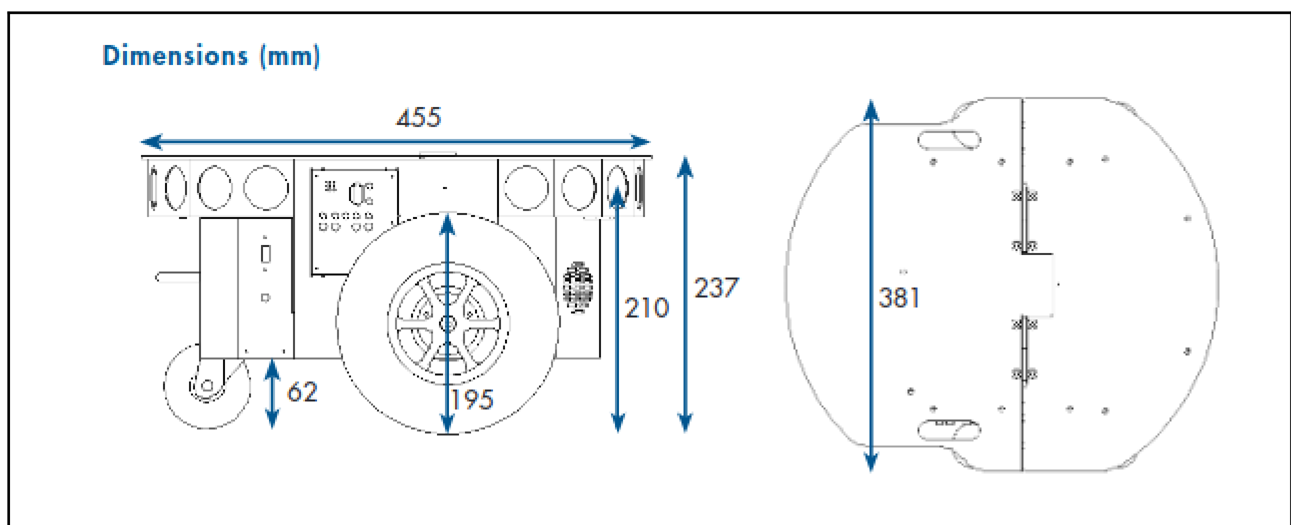



Figura 3.1: Medidas del robot.

De toda la documentación tendremos que tener muy presente las dimensiones del robot de cara a preparar los escenarios de entrenamiento y los lugares en los que le vamos a dar uso en la práctica,

prestaremos atención a los movimientos que puede realizar, que son girar sobre sí mismo, avanzar, retroceder, y girar y avanzar o retroceder simultáneamente.

También deberemos tener en cuenta el tipo de sensor. El sensor es una pieza añadida, por lo que no la encontraremos directamente en la documentación. En nuestro caso vamos a utilizar un láser infrarrojo que funciona en forma de sónar: lanzará rayos a 5 metros de distancia en forma de sector circular mayor de 120 grados, dividirá este ángulo en 683 muestras (esto es por la definición del sensor en software) y por cada punto de estas nos dirá a qué distancia ve los objetos que estén en su radio, si es que están. Si no están en el radio, indicará un valor predefinido como infinito.

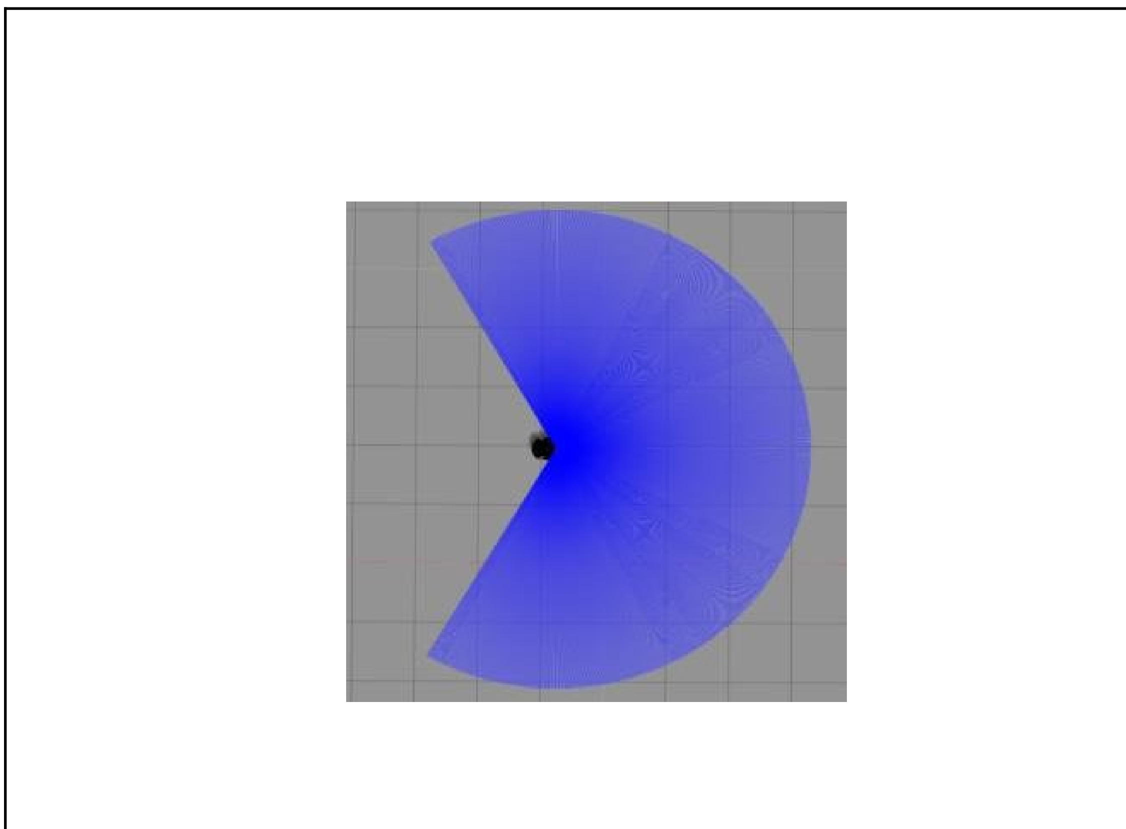


Figura 3.2: Vista cenital que representa el sensor del robot.

Con esto podemos concluir que necesitaremos un código inicial que nos permita:

-Importar el modelo del robot al entorno, con toda su representación real y que incluya integrado el sensor que utilizaremos.

-Obtener información del sensor.

-Ser capaces de darle órdenes al robot para que se mueva en cualquiera de las direcciones reales.

-Ser capaces de saber cuándo se ha chocado nuestro robot.

# 4. Preparación y desarrollo del proyecto.

En esta sección se usarán las tecnologías antes mencionadas para construir nuestro proyecto inicial con el que podremos trabajar e ir haciendo las mejoras necesarias para llegar al objetivo final.

## 4. 1. Familiarización y preparación del entorno.

Para el desarrollo de este Trabajo de Fin de Grado uno de los aspectos transversales más importantes y que más problemáticas puede causar por las distintas tecnologías que se usan es la compatibilidad entre estas.

En primer lugar, respecto a nuestra máquina [Tabla 4.1], se ha optado por no hacer uso de máquinas virtuales, ya que este proyecto hace uso de aprendizaje automático, por lo que las demandas de CPU y GPU pueden ser demasiado altas como para hacer uso de una. Se recomienda usar la opción que más recursos nos otorgue así que incluso si se tiene la posibilidad de utilizar algún sistema distribuido que nos permita realizar simulaciones en paralelo para entrenar al sistema podría ser de gran utilidad para mejorar los tiempos.

| **Componente** | **Especificación** |
|---|---|
| Procesador | AMD Ryzen 5 3600 6-Core Processor |
| Tarjeta gráfica | GP107 [GeForce GTX 1050 Ti] |
| Placa base | B450 AORUS PRO-CF |

| Memoria RAM | 16GiB |
|---|---|

Tabla 4.1: Especificaciones máquina local

Con el fin de trabajar con ROS lo más conveniente es usar alguna distribución de Linux, concretamente una de Ubuntu. Ya que vamos a trabajar con la distribución de **Ros-melodic** [2] por su estabilidad y porque es relativamente nuevo, la versión que funciona con ella es **Ubuntu 18.04** [5]. Si necesitamos usar otra versión, deberíamos ver qué Ubuntu es compatible en la propia documentación de ROS, pero en general se recomienda hacer uso de las mismas configuraciones y versiones que se usarán a lo largo de esta memoria.

Figura 4.1: Distribuciones ROS

La visualización 3D y de las físicas ROS hace uso de **Gazebo** [3], aplicación que nos será muy útil para ver qué está ocurriendo en todo momento, pero en el futuro será conveniente que deshabilitemos la visualización para incrementar la velocidad de cómputo de cara al entrenamiento.

En caso de necesitarlo, para facilitar la instalación de ROS y Gazebo se ha incluido una referencia a un tutorial [6] al final del documento que nos ayudará a realizar la instalación de ROS y de todos sus paquetes.

El lenguaje de programación principal del que se hará uso y se recomienda es **Python** [7] en su versión 2.7 que junto a los paquetes instalados por ROS y las librerías más comunes para desarrollar

aprendizaje automático conformarán nuestro entorno de desarrollo principal.

## 4. 2. Lista de comandos ROS frecuentemente utilizados.

Aquí se muestra una tabla con una recopilación de comandos que se utilizarán muy frecuentemente a lo largo del desarrollo de un proyecto ROS con el fin de facilitar la búsqueda a futuro de estas utilidades sin tener que recurrir a la documentación oficial. Estos comandos se irán mostrando con su uso lógico cronológicamente a lo largo del propio apartado de “Preparación y desarrollo del proyecto”.

| Comando | Descripción | Ejemplo de uso |
|---|---|---|
| $catkin_make | Creación de directorio de trabajo catkin. Necesario para crear y modificar paquetes cakin. | |
| $source | Modifica el directorio de trabajo ROS. | $source catkin_dir/devel/setup.bash |
| $catkin_create_pkg | Creación de paquete catkin | $catkin_create_pkg my_package |
| $roslaunch | Inicio de launchfile | $roslaunch my_package file.launch |
| $rostopic list | Mostrar tópicos disponibles. | |
| $rostopic echo | Mostrar salida de la suscripción a un tópico publicado. | $rostopic echo topic_name |

Tabla 4.2: Lista de comandos

## 4. 3. Directorio de trabajo Catkin.

Con el fin de poder crear nuestros propios paquetes para ROS tendremos que crear un **directorio de trabajo de catkin** o un **catkin workspace**. Esto nos permitirá crear y modificar paquetes catkin, estructurar mejor nuestro proyecto ROS, construirlo e instalarlo en el futuro.

Catkin estará ya instalado en nuestra máquina gracias a la instalación de ROS, pero por si

cualquier cosa no estuviera instalado, podemos hacerlo manualmente. A continuación, crearemos nuestro directorio catkin:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

Tras hacer esto, en nuestro directorio se crearán dos directorios extra, devel y build. En devel tendremos varios ficheros setup.*sh, que si ponemos como fuente mediante el comando “$ source *.sh” a cualquiera de ellos se harán que el directorio catkin sobre el que estamos trabajando se añada como directorio de trabajo para ROS.

```
$ source devel/setup.bash
```

Y si accedemos a la variable del sistema en la que están nuestros paquetes de ROS obtendremos lo siguiente

```
$ echo $ROS_PACKAGE_PATH
/home/youruser/catkin_ws/src:/opt/ros/melodic/share
```

# 4. 4. Creación de paquete catkin

Para crear nuestro paquete haremos uso del comando:

```
$ cd catkin_ws/src
$ catkin_create_pkg pioneer3dx
```

En este paquete contendremos ficheros de configuración, ficheros de lanzamiento, código

fuente, los mundos de entrenamiento, y cualquier otro tipo de información relevante para el proyecto, por lo que se podría decir que será nuestro entorno de trabajo más importante. Además, a priori no tendremos dependencias de otros paquetes, pero en caso de necesitarlas podemos añadirlas.

El siguiente paso será configurar nuestro fichero xml con metainformación del paquete, package.xml. No entraremos en gran detalle en qué representa cada línea, pero se explica bastante por sí mismo:

```xml
<?xml version="1.0"?>
<package format="2">
  <name>pioneer3dx</name>
  <version>0.0.0</version>
  <description>The pioneer3dx package for Manuel Andrés Carrera Galafate's TFG</description>

  <maintainer email="alu0101132020@ull.edu.es">Manuel Andrés Carrera Galafate</maintainer>

  <license>BSD</license>

  <url type="website">https://github.com/maancga</url>

  <buildtool_depend>catkin</buildtool_depend>

</package>
```

Figura 4.2: Contenido de package.xml.

Para finalizar, volveremos a ir a la ruta $ ~/catkin_ws/src, haremos uso del comando catkin_make, y una vez se compile el paquete, volveremos a añadir el directorio de trabajo mediante el comando source. Con esto podremos dar por finalizada la creación de la estructura básica del proyecto.

# 4. 5. Primera ejecución de Gazebo

A continuación el objetivo será crear un entorno de simulación vacío donde podamos ver nuestro robot para familiarizarnos con el entorno de Gazebo.

```xml
<launch>


   <!-- Gazebo  -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch" />

  <!-- Robot with diff drive -->
  <include file="$(find multi_robot_scenario)/launch/pioneer3dx.gazebo.launch">
     <arg name="robot_name" value="r1" />
     <arg name="robot_position" value="-x 0.0 -y -0.5 -z 0.01 -R 0 -P 0 -Y +1.57" />
   </include>

  <node pkg="tf" type="static_transform_publisher" name="base_to_odom_r1"
     args="0.0 -0.0 0.0 0.0 0.0 0.0 base_link r1/odom 200" />

  <!-- RVIZ -->
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find gazebo_plugins)/test/multi_robot_scenario/launch/pioneer3dx.rviz'

</launch>
```

Figura 4.3: Código de ejemplo de un launchfile que contenga un mundo vacío y nuestro robot.

ROS incluye una serie de comandos para terminal entre los que se incluye **roslaunch**. Este comando nos permitirá arrancar launchfiles, ficheros con extensión .launch que incluirán información en formato XML. Para ver en detalle la formación de los ficheros podremos ver la documentación de la wiki de ROS [8], pero básicamente tenemos que saber que desde estos ficheros podemos arrancar otros launchfiles e incluso scripts. En primer lugar hemos optado por crear un directorio de nombre /launch dentro del paquete que hemos creado, y dentro de este hemos creado un fichero de arranque que instancie un mundo vacío, a continuación cargue nuestro robot. Además hemos cargado el paquete tf y el paquete rviz para poder visualizar cierta información sobre el robot, pero esto no es realmente relevante (todos estos paquetes vienen ya incluidos en la instalación normal de gazebo). Una vez el launchfile esté listo, tendremos que ejecutar el comando $ roslaunch pioneer3dx nuestro_fichero.launch y con ello deberíamos ver el siguiente resultado:

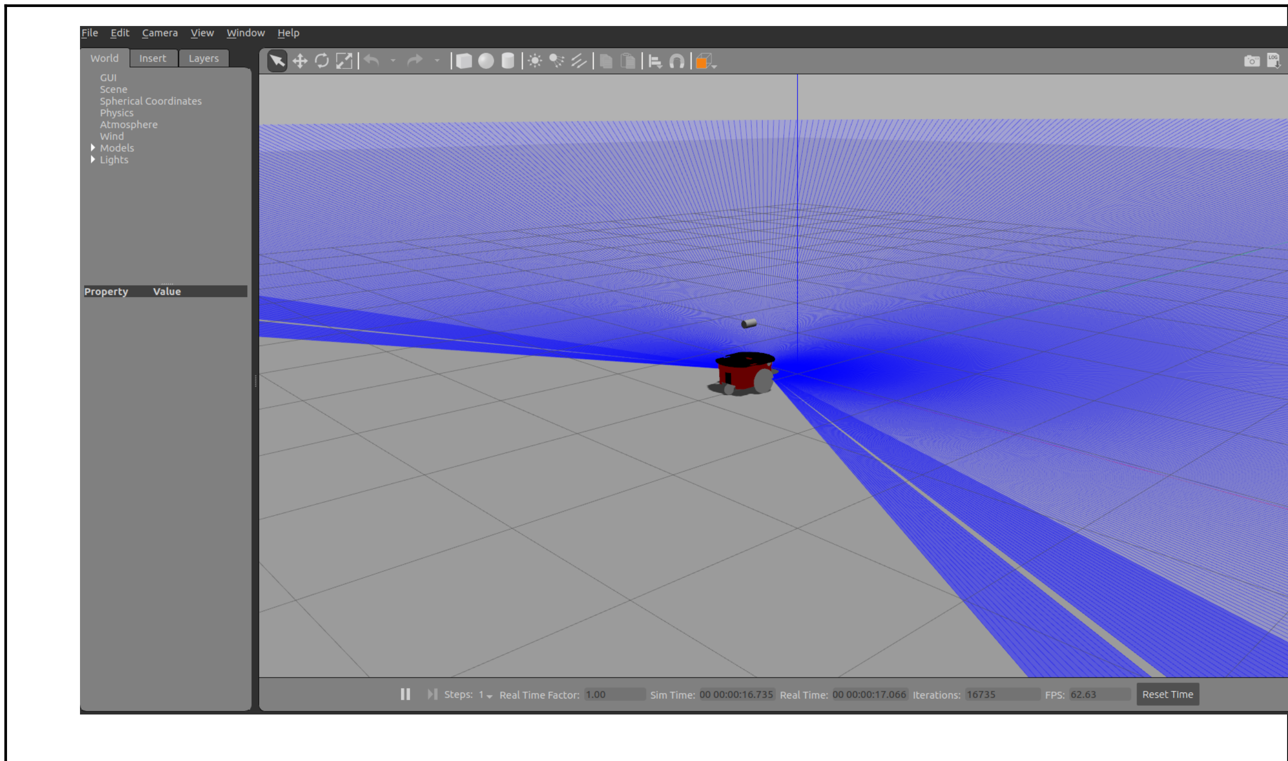



Figura 4.4: Gazebo con robot y mundo vacío

# 4. 6. Tópicos ROS.

Llegados a este punto somos capaces de cargar un escenario y un robot sin mucho problema, pero esto no es de gran utilidad si no podemos comunicarnos con el entorno y especialmente si no somos capaces de hacerlo de una manera programática. ROS nos provee con el sistema de tópicos para lograr la comunicación entre distintos elementos del entorno.

Un tópico es un canal de comunicación para publicadores y suscriptores, donde un publicador es un emisor de mensajes y un suscriptor será un receptor. La tecnología que utiliza Gazebo internamente es de sockets TCP/IP, así que en esencia los tópicos son una interfaz para ellos. Toda la comunicación se basa en la emisión y recepción de mensajes, que son un tipo de estructuras de datos concreta. Dependiendo del tópico al que nos queramos suscribir, o del tipo de información que deseemos emitir tendremos que adaptarnos a estas.

Continuando con el ejemplo anteriormente ejecutado, al iniciar nuestro robot, automáticamente se publicarán una serie de tópicos. Estos serán visibles mediante el uso del comando $rostopic list:

```
/r1/cmd_vel
/r1/front_camera/camera_info
/r1/front_camera/image_raw
/r1/front_camera/image_raw/compressed
/r1/front_camera/image_raw/compressed/parameter_descriptions
/r1/front_camera/image_raw/compressed/parameter_updates
/r1/front_camera/image_raw/compressedDepth
/r1/front_camera/image_raw/compressedDepth/parameter_descriptions
/r1/front_camera/image_raw/compressedDepth/parameter_updates
/r1/front_camera/image_raw/theora
/r1/front_camera/image_raw/theora/parameter_descriptions
/r1/front_camera/image_raw/theora/parameter_updates
/r1/front_camera/parameter_descriptions
/r1/front_camera/parameter_updates
/r1/front_laser/scan
/r1/joint_states
/r1/odom
```

Figura 4.5: Lista de tópicos disponibles

Cada uno de estos tópicos tiene una finalidad distinta. Por ejemplo, el tópico "/r1/front_laser/scan" nos brindará información constante de la detección del sensor láser del robot 1, mientras que "/r1/odom" nos mostrará información sobre la posición actual de nuestro robot. Para poder ver en una terminal alguna de esta información, podremos usar el comando $ rostopic echo /nombre/del/tópico. Hay que tener en cuenta que cuando nos suscribimos a estos tópicos recibiremos un flujo constante de mensajes mientras el tópico siga publicado o nosotros sigamos suscritos.

Los tópicos también son accesibles desde programas. El paquete de ROS nos brinda algunas funciones para directamente desde un programa en Python poder obtener información de un tópico o incluso enviar información. Por ejemplo, si ejecutamos aun con nuestro robot en el mundo vacío, en una nueva terminal, el comando:

$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py cmd_vel:=r1/cmd_vel

Se ejecutará un programa que nos permitirá enviar información al robot y cambiar su pose, pudiendo avanzar, retroceder, girar, o hacer que no se mueva.

Para crear nuestros propios programas que nos permitan interactuar con el entorno ROS

haremos uso de las funciones Subscriber y Publisher.

La función Subscriber necesitará como parámetros el tópico del que quiere recibir información, qué tipo de mensaje va a recibir y una función de callback que se ejecutará cada vez que llegue un mensaje.

La función Publisher como parámetros recibirá el tópico al que enviará información, el tipo de mensaje que enviará, y el tamaño de mensaje que se usará para los envíos que serán asíncronos.

```
rospy.Subscriber("/r1/odom", Odometry, self._odom_callback)
rospy.Subscriber("/r1/front_laser/scan", LaserScan, self._laser_scan_callbac

self._cmd_vel_pub = rospy.Publisher('/r1/cmd_vel', Twist, queue_size=1)
```

Figura 4.6: Ejemplo de suscripción y publicación de tópicos.

Con todas estas herramientas ya deberíamos ser capaces de arrancar un escenario con nuestro robot cargado y arrancar un programa que nos permita mover el robot de manera automática.

# 5. Aprendizaje automático y tratamiento de datos.

Nuestro proyecto va a hacer uso de aprendizaje automático para lograr el objetivo de tener un robot móvil inteligente. Dependiendo del tipo de datos con el que vamos a trabajar hay arquitecturas de aprendizaje que son mejores que otras para las tareas designadas. Por ejemplo, si nuestro robot dispusiera de una cámara como sensor y trabajase con imágenes, el tipo de arquitectura más adecuado para trabajar sería una red neuronal convolucional.

En nuestro caso, el sensor es un láser que chocará contra los objetos del entorno y nos retornará información de los distintos puntos del láser. A un nivel más alto de abstracción, vamos a trabajar con un vector de elementos que representa la distancia en metros a los distintos objetos que se detecten en el radio del sensor, y nuestro objetivo será elaborar una respuesta inteligente a la información que represente esta estructura de datos. Así por ejemplo, el siguiente vector:

[Infinito, Infinito, 5.1, 4.2, 3.0, 0.5, 3.0, 4.2, 5.1, Infinito, Infinito]

Sería la lectura de un sensor láser de 10 puntos, donde no detecte ningún objeto por los extremos (de ahí que marque infinito, ya que la distancia del láser es limitada y de esta manera podemos representar que no se está detectando ningún objeto), y a medida que los elementos son más centrales (lo que sería la parte delantera del robot) más cerca estarían. Esto por ejemplo podría ser la entrada del sistema de detección para objetos que tenga la forma de la esquina de una pared, una pirámide, etc...

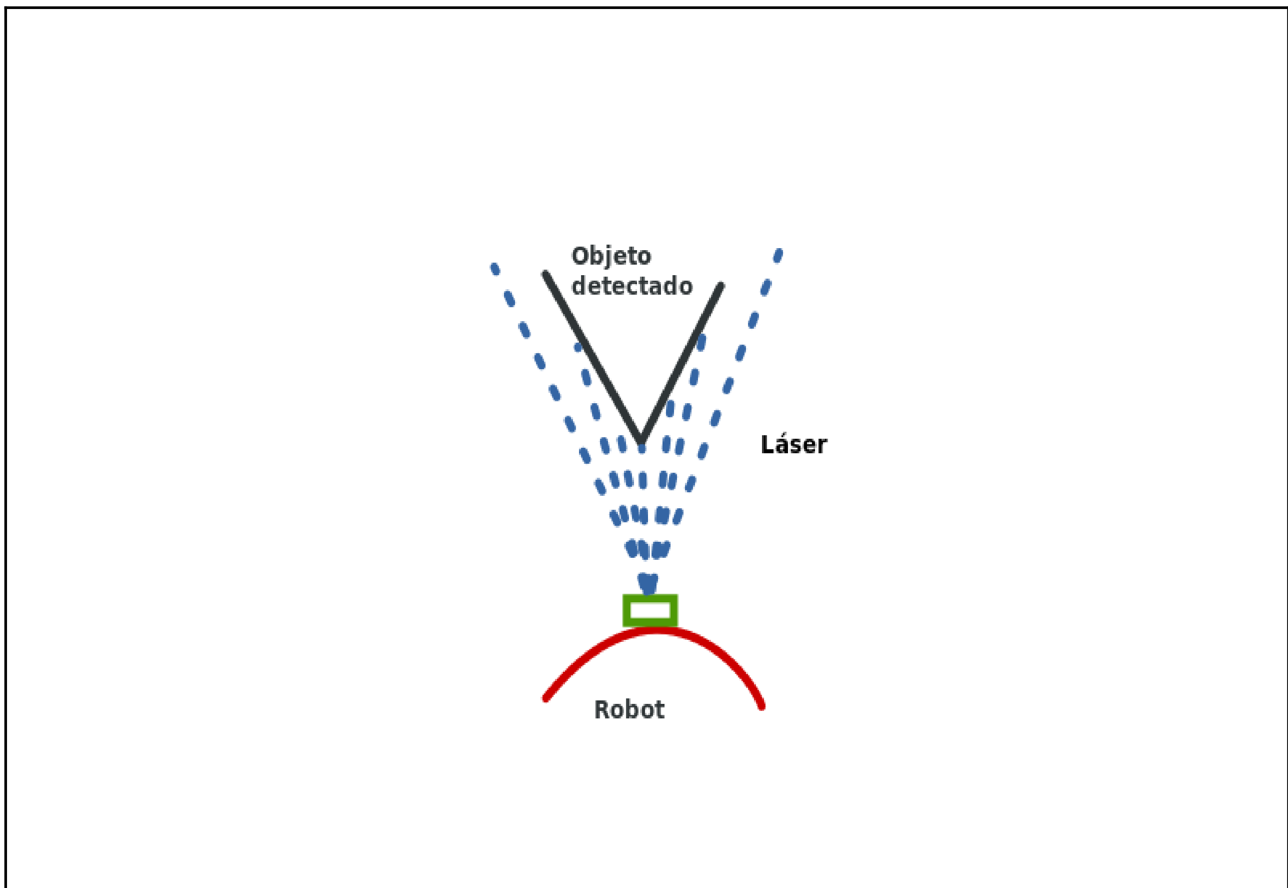



Figura 5.1: Diagrama representativo del vector de ejemplo.

Para el caso representado en la figura [Figura 5.1] la respuesta que esperaríamos de un robot inteligente sería que girase en lugar de avanzar.

Un matiz muy importante a tener en cuenta es que nuestro sensor tiene forma de plano, es decir, solo representa coordenadas en los ejes X e Y, por lo que si hay objetos más altos o más bajos que la altura del sensor, estos van a ser obviados. A pesar de que esto pueda parecer un inconveniente porque perdemos información, si trabajamos en entornos donde sabemos que todos los objetos que pueda chocar nuestro robot están en una altura adecuada para el uso del sensor o esta es regulable de alguna manera, simplificaremos mucho los cálculos y ganaremos mucha eficiencia en el entrenamiento.

Otro aspecto a tener en cuenta es que nuestro robot en la simulación se detecta a sí mismo por los extremos, y dado el tamaño del vector (más de 600 elementos), a que estos datos siempre se van a repetir, que las acciones no dependen de ellos y al gran número de cálculos e iteraciones que se van a realizar, hemos eliminado del entrenamiento estos datos, ya que podrían considerarse como ruido.

## 5. 1. Q-Learn

Q-Learn es el primer algoritmo que utilizaremos para entrenar por su sencillez.

"Q-learning es una técnica de aprendizaje por refuerzo utilizada en aprendizaje automático. El objetivo del Q-learning es aprender una serie de normas que le diga a un agente qué acción tomar bajo qué circunstancias. No requiere un modelo del entorno y puede manejar problemas con transiciones estocásticas y recompensas sin requerir adaptaciones."

"El aprendizaje por refuerzo involucra a un agente, un conjunto de estados S, y un conjunto A de acciones por estado. Llevando a cabo una acción a ∈ A, el agente pasa de un estado a otro. a ∈ A La ejecución de una acción en un estado concreto le proporciona una recompensa al agente (una puntuación numérica).

El objetivo del agente es maximizar su recompensa total. Lo hace sumando la máxima recompensa alcanzable, en estados futuros, a la recompensa por alcanzar su estado actual, influyendo eficazmente en la acción actual por la futura recompensa potencial. Esta recompensa potencial es una suma ponderada de la esperanza matemática de las recompensas de los futuros pasos empezando desde el estado actual." [9]

Es un algoritmo sencillo, que solo funciona para entradas de tipo discreta, pero que para casos como el nuestro son potencialmente útiles, dándonos a priori una buena solución sin una gran dificultad de implementación.

# 6. Desarrollo python.

ROS nos permite trabajar en C++ y Python, en nuestro caso nos hemos quedado con la segunda opción por varios motivos: Python es la opción preferida en los desarrollos basados en aprendizaje automático, en primer lugar por su pronunciada curva de aprendizaje, muy conveniente para un lenguaje que no se usa solo por informáticos o personas que se dediquen exclusivamente a la programación, sino otros muchos perfiles orientados a la ciencia, como pueden ser físicos, matemáticos o ingenieros de distintas ramas. Otro motivo es que dispone de un muy adecuado ecosistema de librerías dedicado al estudio y representación de información: **pandas, tensorflow, Matplotlib, Keras, gymAI…** que son ideales para proyectos de esta naturaleza. Y además, uno de los puntos esenciales, la facilidad de instalación de todas los módulos, independientemente del entorno en el que se trabaje.

## 6. 1. Gym AI.

**GymAI**. es la librería principal con la que trabajaremos para la codificación del aprendizaje reforzado. Dispone de una gran documentación, y es frecuentemente usada con ROS para la formalización de robots que utilizan aprendizaje automático.

“Gym es un conjunto de herramientas para desarrollar y comparar algoritmos de aprendizaje por refuerzo. Permite enseñar a los agentes todo tipo de cosas, desde caminar hasta jugar a juegos como el Pong o el Pinball.” [10]

Para poder utilizar las herramientas de gym en casos personalizados como el nuestro, debemos crear nuestros propios entornos (environments). Necesitaremos un entorno que refleje la realidad del simulador Gazebo, que sea capaz de interactuar con él, y calcular recompensas adecuadas para entrenar a nuestro sistema.

## 6. 2. Estructura de clases de los robots.

Hemos seguido este tutorial [11] para crear nuestro entorno. Hemos cogido como base los entornos tanto de robot como de tareas del paquete openai_ros y lo hemos adaptado a nuestro propio robot, el Pioneer3dx. Hay que tener en cuenta que la estructura de clases que hemos generado es la mostrada en la imagen [Figura 6.1]

En primera instancia puede parecer un tanto abrumador, pero intentaremos explicarlo por partes.

**Gym env:** Esta clase es la interfaz dispuesta por la librería Gym para crear nuestros propios entornos que nos servirán para entrenar. Los métodos y propiedades más relevantes, y que tendremos que sobrecargar en las clases hijas, son las mostradas en el diagrama UML [Figura 6.1].

- En **action_space, observation_space y reward_range** tendremos que definir el tipo de entrada del entorno y su dominio, es decir, si es discreto o continuo, qué valores puede tomar, etc.
- **Step** definirá las implicaciones en el entorno de la acción que haya escogido el agente en un instante en concreto.
- **Reset** definirá cómo se reinicia un episodio de entrenamiento de nuestro agente.
- **Render** nos dirá cómo deberá representarse el agente en un instante en concreto. En nuestro caso esta característica no se usará ya que la representación del robot la veremos a través de Gazebo.
- **Close** es donde definiremos acciones que debamos realizar para limpiar el entorno ahí donde no se haga de manera automática.
- **Seed** es donde definiremos cómo se genera la semilla que dictará los resultados de la aleatoriedad.

**Robot Gazebo env:** Esta clase es la interfaz dispuesta por el paquete openai_ros, que definirá la base para cualquier robot gazebo en un entorno gym heredando de esta misma clase, y nos permitirá más adelante crear las instancias para robots concretos. La parte de las propiedades no debe preocuparnos mucho, ya que no son muy relevantes más allá de declaraciones de algunas variables y utilización de algún elemento de conexión que no detallaremos en el TFG. Respecto a los métodos que sobreescribe de su clase padre, básicamente los adapta al entorno ROS, haciendo alguna pausa cuando es necesario, alguna asignación y otras instrucciones que se podrán ver en detalle en la documentación de openai_ros (Referencia).

Figura 6.1: Diagrama UML de las clases principales involucradas para representar el robot.

**Pioneer3dx env:** Esta clase hereda del entorno de robot gazebo, y aquí es donde definiremos la suscripción y el uso de nuestros sensores, cómo moveremos a nuestro robot, cómo sabremos si ha chocado o no y cualquier otra cuestión necesaria para definir un agente que tenga las capacidades de nuestro robot real en el entorno que hemos definido.

**Pioneer3dx task:** Esta clase que hereda de nuestro entorno Pioneer3dx es la que definirá el contexto de la tarea que se quiere realizar y el robot que utilizaremos. Esta separación nos permite adaptar el código que hagamos de una forma más modular a posibles cambios en los robots, en los algoritmos, o en el conjunto de acciones que queremos que pueda realizar nuestro robot.

Por ejemplo, en la clase Pioneer3dx hemos definido cómo se puede mover el robot a partir de una velocidad angular y lineal, pero en nuestra clase de mayor nivel, definiremos que el robot solo puede realizar 3 tipos de movimiento, giro a la izquierda, giro a la derecha y avanzar, y definiremos qué implica eso a menor nivel. Así hemos definido un contexto en el que tenemos un robot que puede realizar 3 tipos de acciones, sin tener que modificar la definición de nuestro Pioneer3dx.

# 6. 3. Aprendizaje reforzado.

Finalmente, una vez hemos definido nuestro robot, que es capaz de moverse, nos queda utilizar algoritmos que enseñen a este a moverse de manera inteligente. En primer lugar hemos tratado de implementar un algoritmo Q-Learn por su sencillez y para comprobar que todo está funcionando de manera correcta. Como hemos dicho anteriormente, no entraremos en los detalles de la implementación, pero hablaremos del diseño del aprendizaje por el que se ha optado.

En primer lugar, hemos hecho una simplificación a la observación de los datos. Dado a la gran cantidad de puntos que recibe el láser y lo mucho que podría incrementar el espacio de estados posibles para un agente, hemos optado por hacer una simplificación. Hemos definido un valor **n** que será el número de muestras totales que se usarán para representar a todo el conjunto de datos. Cada una de estas n muestras representará a m puntos definidos por la ecuación:

$$m = \frac{\textit{número de muestras totales}}{n}$$

Cada m puntos del vector que retornará el sensor representará uno de los n valores. Así por ejemplo para 10 valores totales, con una n igual a 5, m será 2, por lo que los valores que recibirá el robot como entrada serán el primero, el tercero, el quinto, el séptimo, y el noveno.

Detección real: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Simplificación: [1, 3, 5, 7, 9]

Para nuestro caso, experimentalmente estamos probando con un n igual a un número en el rango [6-10].

Esta simplificación la hacemos porque consideramos que los entornos donde estamos trabajando contienen objetos lo suficientemente grandes como para que no se pierda gran información y se acelere mucho el cómputo, ya que no estamos procesando más de 600 elementos que antes sí.

Otra simplificación que hemos hecho es discretizar todos los valores de este vector resultado, ya que de no hacerlo, volveríamos al mismo problema en el que tendríamos un espacio demasiado grande de valores, que para un algoritmo de tipo Q-Learn por ejemplo supone un gran problema.

Con todo esto ya podemos empezar a ejecutar algoritmos diversos y empezar a obtener resultados.

# 7. Problemáticas y posibles soluciones.

Cuando se trabaja con tantos datos y en un entorno tan complejo como el de una simulación física hay muchos problemas que se pueden dar cuando los datos que representan la realidad son erróneos y dan a entender situaciones que realmente no están ocurriendo. Todo esto se suma a las problemáticas ya de por sí frecuentes que se dan cuando se trabaja con sistemas de aprendizaje automático en lugar de con algoritmos clásicos. A lo largo de este apartado veremos estas incidencias y posibles soluciones que se han planteado durante el desarrollo, con el objetivo de que el lector pueda prevenir algunas de las situaciones indeseadas que se verán en el apartado de resultados.

## 7. 1. Problemas de la simulación.

En primer lugar, el láser que teníamos inicialmente instalado en nuestro robot tiene un error, y es que cuando giramos, en algunos instantes detecta objetos cercanos que realmente no existen. Esto es un problema ya que al detectarlo cerca, en muchas ocasiones tras girar, la simulación se reinicia ya que piensa que ha chocado. Para solucionar este problema, tuvimos que acceder al fichero de definición del robot y cambiar el láser que estaba usando del que trae por defecto a uno GPU que no contiene este error.

Tras arreglar este primer problema, nos encontramos con otro bastante grave. El robot en ocasiones al aproximarse demasiado a una pared hace que el láser atraviese el muro antes de detectar que ha chocado. Esto hace que haya datos falsos, ya que en ocasiones se está reiniciando recibiendo la información de que tiene objetos delante que están muy lejos. Además, como el nuevo láser que usamos tiene un radio bastante superior al que usábamos anteriormente, el error es aún más grande, porque detecta objetos hasta 40 metros de distancia, lo cual hace que el peso de esta variable sea muy influyente y no permite al robot entrenar de manera correcta.

Para ambos problemas especulamos que los problemas puedan estar relacionados con el Hardware utilizado y el procesamiento, por lo que la solución consistirá en ajustar la simulación y

los sensores que estamos utilizando a unos que se adecuen mejor a el hardware que estamos utilizando, o como alternativa manipular el código de tal manera que obviemos estas situaciones que en el entorno real no se darían. Por ejemplo, para el atravesado de paredes, si detectamos un elemento a 40 metros del robot cuando este ha chocado, podemos tratarlo como uno que esté a 0, o para el error de las detecciones en giros, podemos averiguar a qué distancia se están detectando estos objetos, y ajustar la distancia mínima de choque a una lo suficientemente pequeña como para que se obvien estos objetos.

Finalmente respecto a ambas problemáticas hemos optado por mantener el primer láser con la solución por Software de detectar estos casos de "objetos fantasmas" y obviarlos en base a la distancia a la que detecta objetos el robot.

## 7. 2. Problemas de diseño del aprendizaje reforzado.

Un error que podemos encontrar si no ajustamos correctamente las recompensas es que si no se da la suficiente retroalimentación positiva a avanzar respecto a girar, podemos encontrar situaciones en las que nuestro agente prefiera girar sobre sí mismo a continuar avanzando. Obviamente la solución a esto será puntuar mejor los avances respecto a las rotaciones.

Otra problemática a tener en cuenta es la capacidad de cómputo y los tiempos de ejecución. El aprendizaje reforzado presenta cierta incertidumbre sobre los resultados que se mostrarán, si las hipótesis sobre las que se diseña el sistema son correctas, o si la implementación es lo suficientemente buena como para que llegue a buen puerto. Por si esto fuera poco, tenemos que generar y trabajar con una cantidad ingente de datos, que requieren su respectiva computación. El algoritmo por ejemplo Q-learn como podemos ver en la demostración [12] puede tardar en converger y empezar a mostrar una solución que empiece a ser satisfactoria más de 2000 iteraciones, que en tiempo de cómputo para nuestra máquina local [Tabla 4.1], a una velocidad de simulación 10 veces superior a la velocidad real, son más de 2 horas (Esto en el caso de que todo haya ido bien). A lo largo de este TFG se han llegado a realizar entrenamientos de más de 10 horas que no han llegado a aproximarse siquiera a soluciones deseables [Figura 7.1]. Obviamente la mala implementación o mal diseño de los agentes no es intencionado, y en este sentido no se puede hacer mucho más que ganar experiencia trabajando en este ámbito y realizar mejores diseños a futuro. Sin

embargo una solución interesante podría ser, como se sugirió en el apartado 4, hacer uso de un sistema distribuido que pueda tanto brindarnos las ventajas del paralelismo como mayor potencia de cómputo por máquina.

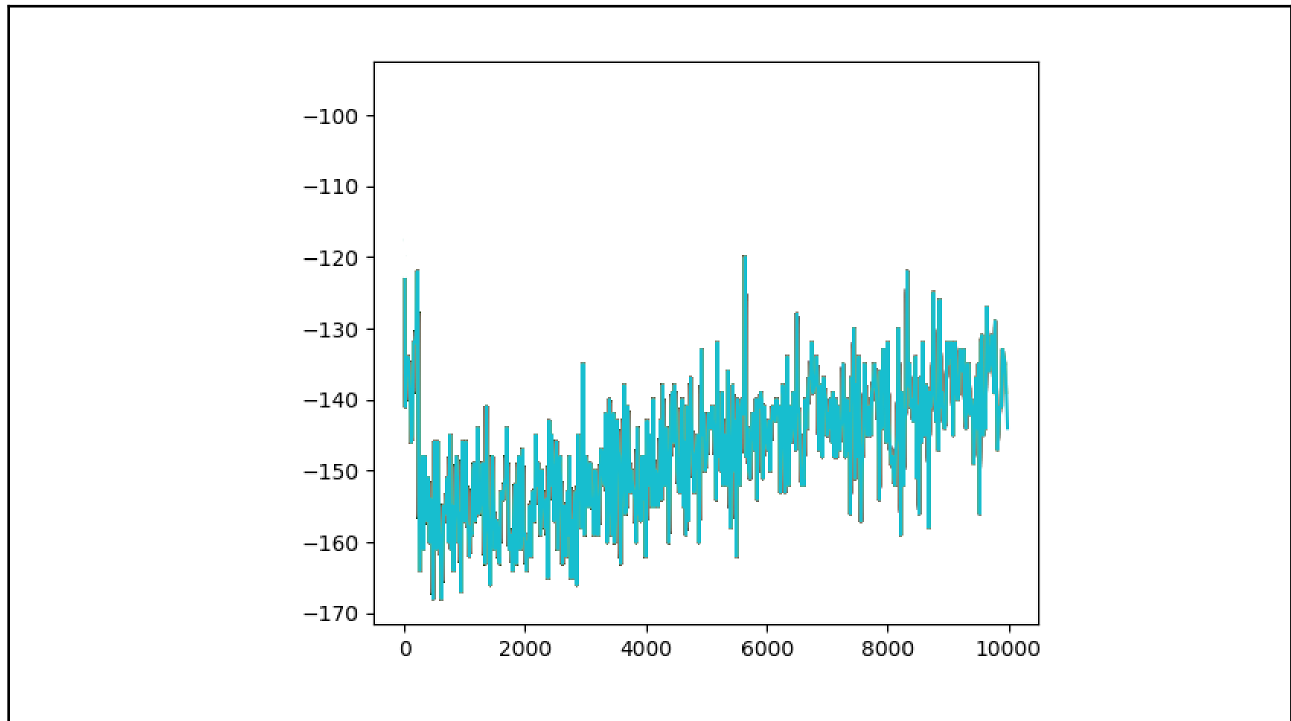



Figura 7.1: Representación ganancia (eje vertical) para para el algoritmo Q-Learn cada 20 iteraciones (eje horizontal).

## 7. 3. Problemas de Software.

Este proyecto combina muchas tecnologías que tienen que ser compatibles entre sí a cada momento, y la obsolescencia, actualización y compatibilidad del código son un problema a considerar. Como ya se mencionó en el apartado 4 de este TFG, se recomienda que en la reproducción del trabajo realizado se sea lo más fiel posible a las tecnologías utilizadas y a los pasos seguidos para obtener resultados similares, o al menos que todos los componentes del sistema funcionen. En nuestro caso aun teniendo tecnologías totalmente compatibles en todo momento, se han encontrado ciertas dificultades al compatibilizar versiones, modificar ramas de repositorios, uso de Hardware no compatible (Nvidia Cuda [13]), y para solucionar esto no se puede hacer nada más que tener mucha precaución con las versiones que se instalan, evitar actualizaciones no deseadas que puedan hacer que el sistema deje de funcionar, y hacer uso control de versiones.

# 8. Experimentación y resultados.

De cara al entrenamiento se ha optado por usar un mapa de forma laberíntica no especialmente complejo [Figura 8.1].

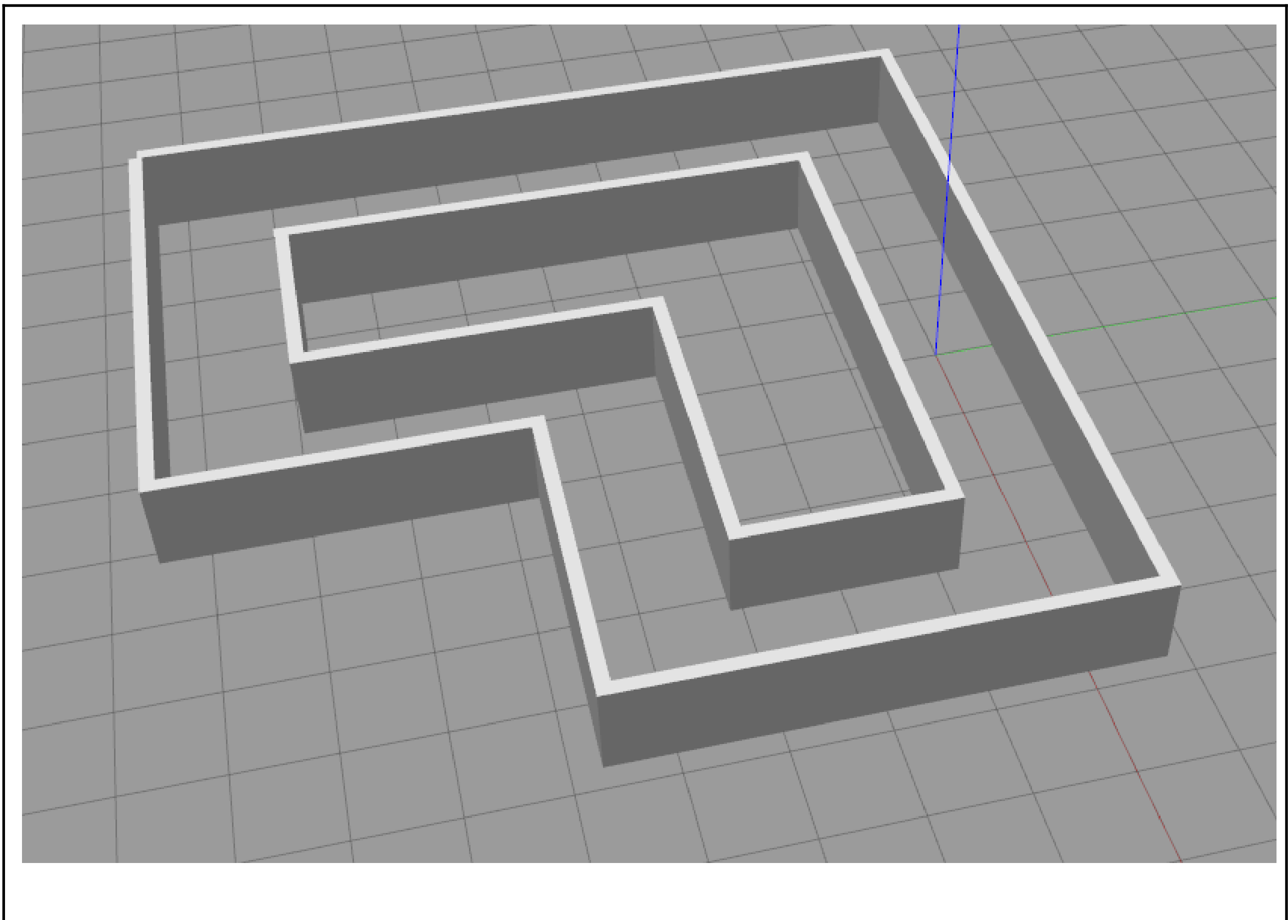


Figura 8.1: Mapa de entrenamiento.

El objetivo inicial es que el robot sea capaz de avanzar por él sin chocarse, y en base al número de movimientos y al tipo de movimiento que haga sin chocarse le daremos una puntuación. Un ejemplo de las puntuaciones asignables sería el siguiente:

- Avanzar -> 5 puntos.
- Girar hacia cualquiera de las 2 direcciones: 1 punto.

- Chocar -> -200 puntos.

El robot en cada iteración recibirá un vector de X elementos (Cada combinación de estos representará un estado para el agente), que son los puntos representativos de los que se habló en el apartado 6.3, y en base a estos datos y las decisiones tomadas previamente, tomará una decisión, recibirá una puntuación asociada a la acción tomada y a partir de ella aprenderá . En la figura 8.2 podemos ver una representación del flujo de aprendizaje para cada iteración.

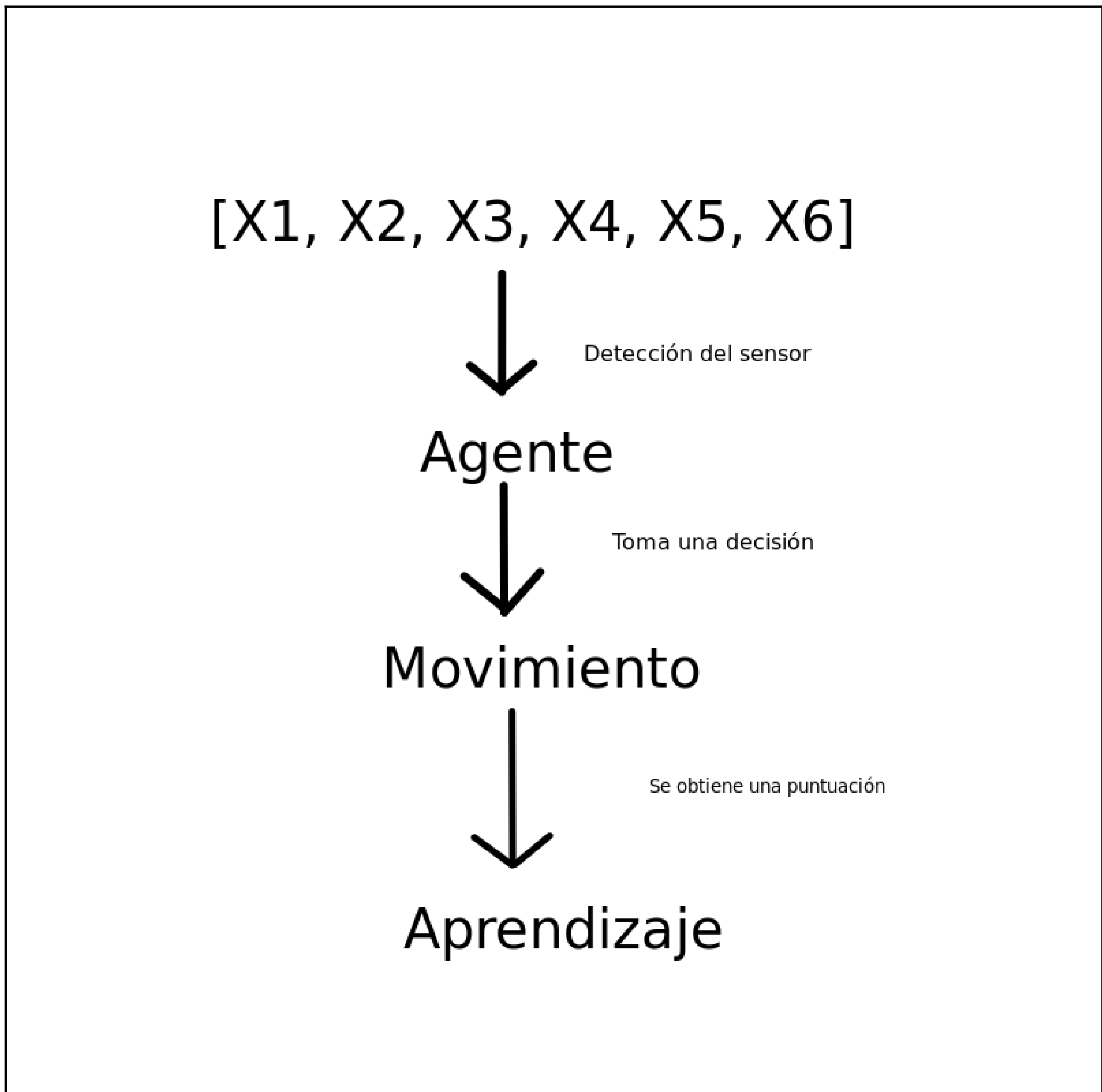
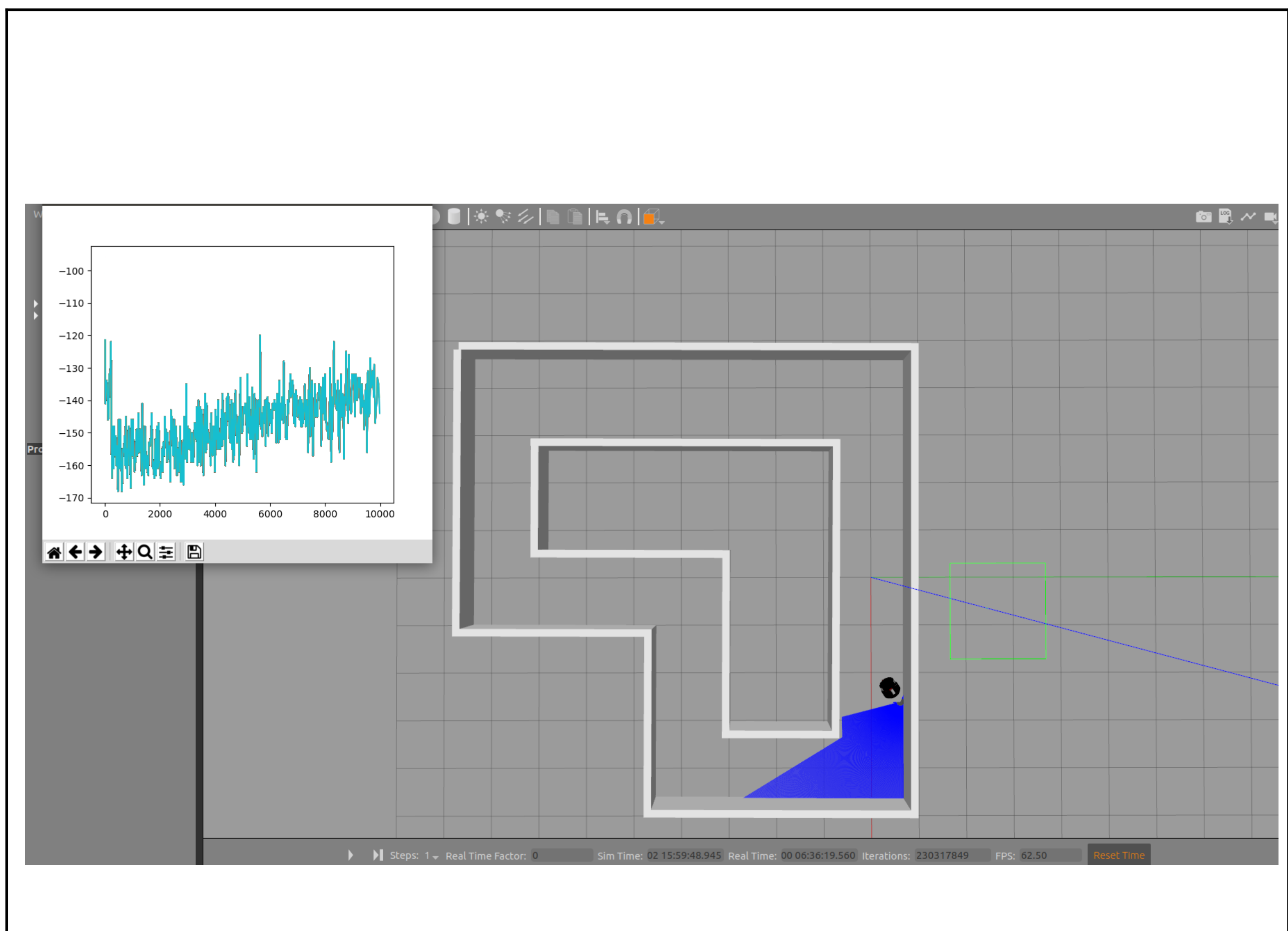



Figura 8.2: Esquema de aprendizaje

Cuando el robot choque, se dará por concluido un episodio, y este proceso se repetirá un número n de iteraciones. Para que se dé esta situación, la condición que se ha tomado es que el robot detecte que alguno de los puntos que está detectando están a una distancia menor a un umbral

M. Esto es importante, porque dependiendo de la configuración del sistema, del Hardware que se use, y de otras cuestiones relacionadas con el simulador físico se pueden dar situaciones inesperadas o indeseadas, como las que se comentan en el apartado 7.1 del TFG.

La principal implementación de aprendizaje reforzado con la que se ha trabajado es el algoritmo Q-Learn, por su sencillez aparente. A lo largo del trabajo se han probado un gran número de configuraciones, alterando los valores de las puntuaciones por movimientos, cambiando el sensor láser de un modelo a otro, alterando la distancia en la que se considera que el Pionner3dx ha chocado, el número de elementos detectados, etc… Pero nunca se ha llegado a obtener una solución que converja: se debería ver en la gráfica de puntuaciones un gran incremento en ciertas secciones, y una estabilización a partir de la cual podemos considerar que el sistema ha encontrado una política con la que pueda intentar maximizar beneficios a partir de la entrada (aunque esta no tenga por qué ser la política óptima).



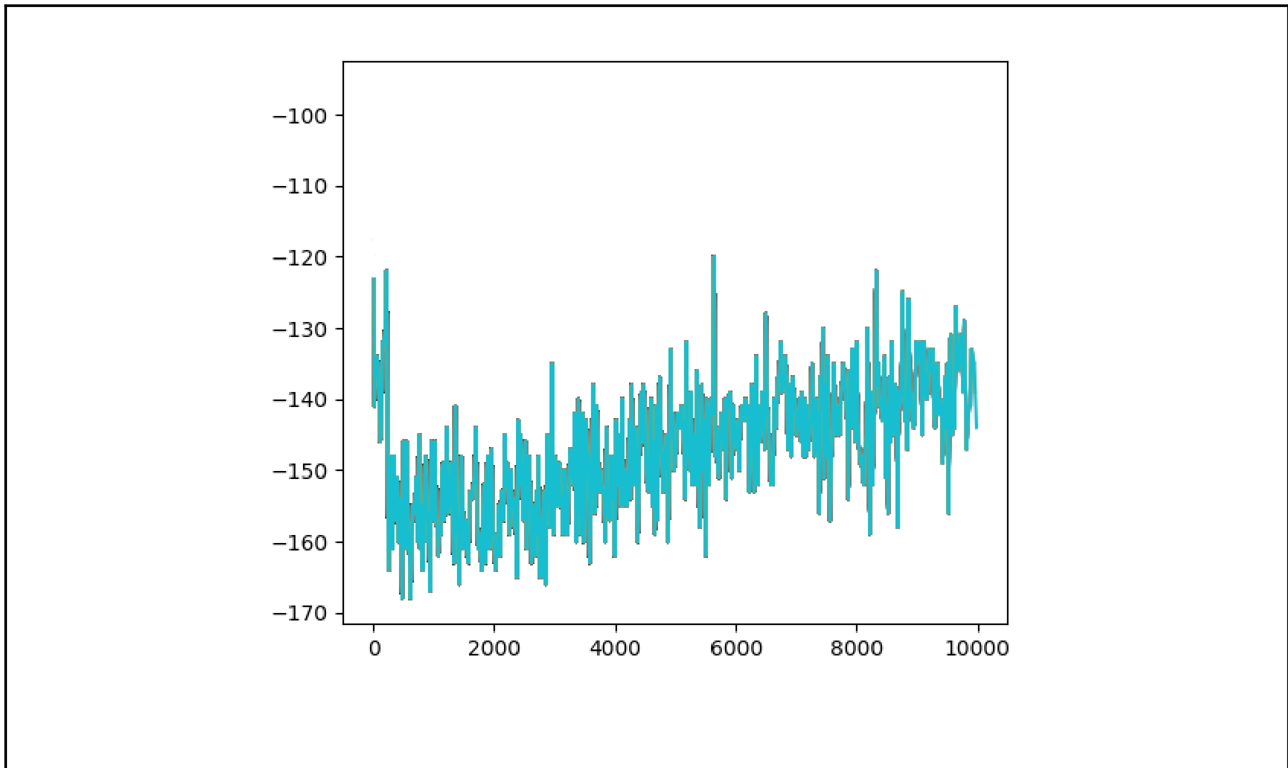



Figura 8.3: Simulador y gráfica de aprendizaje para 10.000 episodios de entrenamiento.

En la figura [Figura 8.3] se puede observar el simulador con su resultado del entrenamiento con la siguiente configuración:

- Número de episodios: 10.000.
- Distancia a la que se considera choque: 0.15 metros (Trabajar con distancias menores acarrea muchos problemas).
- Número de datos representativos que se usarán del láser: 6 puntos.
- Recompensas asociadas a los movimientos:
    - Avanzar -> 5 puntos.
    - Girar hacia cualquiera de las 2 direcciones: 1 punto.
    - Chocar -> -200 puntos.

El tiempo total invertido en este entrenamiento ha sido de 10 horas continuadas en la configuración mostrada en la tabla [Tabla 4.1] a una velocidad de simulación de 10 veces el tiempo real. Como se puede ver en la gráfica, en la que la ganancia promedio viene representada por el eje vertical y el número de iteraciones agrupado en grupos de 20 por el eje horizontal, a pesar de que se puede ver cierta tendencia a la mejora a la larga, los resultados no son claramente concluyentes, o no tan buenos como se pudieran esperar. Para tener un punto de comparación, podemos ver en el

ejemplo de Acutronic Robotics [14], aplicando el mismo algoritmo y en una configuración no muy distinta que se obtienen resultados mucho más afines a lo que se pretende con la experimentación [Figura 8.4]. Aquí no tan solo obtenemos resultados que mejoran sustancialmente a partir ya de 1000 iteraciones, si no que consideramos que el robot llega al objetivo de esquivar, obteniendo puntuaciones muy superiores a la penalización por choque (-200 puntos).

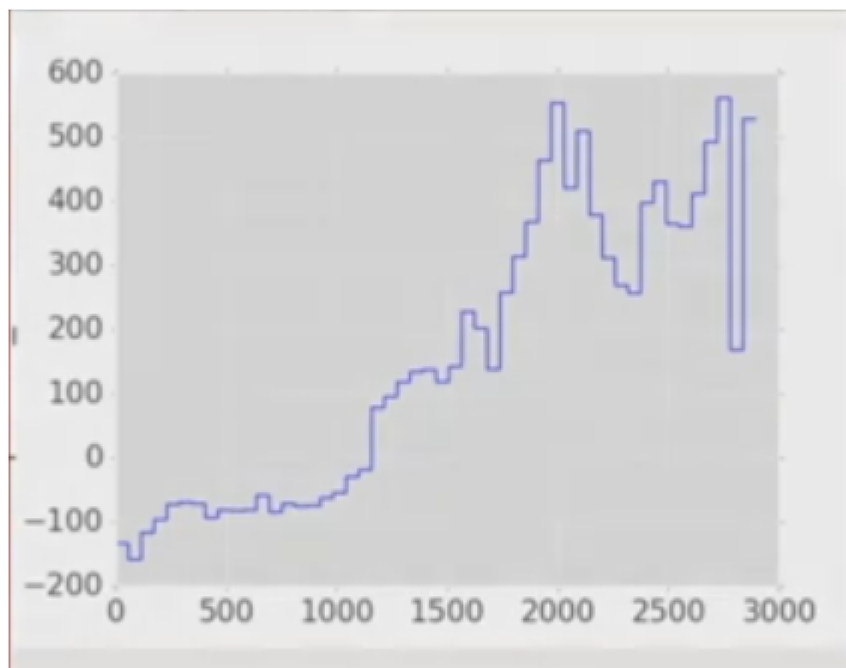



Figura 8.4: Gráfica de aprendizaje del algoritmo Q-Learn de Acutronic Robotics.

Ambos experimentos se basan en el mismo algoritmo y en las mismas librerías para llegar al objetivo, pero hay alguna diferencia en la base que hace que uno obtenga resultados satisfactorios y en el otro caso lo que se obtenga sea algo no muy distinto a lo que realizaría un algoritmo que haga movimientos aleatorios. Incluso en el caso del resultado de Acutronic Robotics, podemos considerar que el problema de la esquiva de obstáculos ha sido resuelto en no más de unas 2 horas (estimando y haciendo una regla de 3 respecto a lo que se ha tardado en entrenar nuestro sistema), un tiempo muy interesante y a tener en cuenta por la facilidad con la que se podría modificar la configuración del algoritmo de aprendizaje, el mapa, o incluso el robot que se está entrenando.

En líneas futuras de trabajo habrá que analizar desde un punto de vista de desarrollo del

Software qué errores pueden estar ocurriendo en la implementación para que el resultado esperado no se de, pero está claro que el algoritmo puede funcionar en este contexto y el uso de las tecnologías usadas para resolver el problema de la navegación es viable. [1]

[1] Los resultados de la parte experimental en el entorno local pueden diferir de aquí al momento de la defensa oral del TFG, se seguirá trabajando en ellos, buscando el origen de las problemáticas en los resultados desde en el propio código hasta a si están relacionados con el Hardware que se está usando y su compatibilidad con el simulador de físicas que se está usando.

# 9. Conclusiones y líneas futuras de trabajo.

Los campos de la inteligencia artificial y la robótica son capaces de brindarnos muy buenas soluciones a un gran espectro de problemas, por lo que son siempre herramientas a considerar en desarrollos de Software. Al final del día son sistemas capaces de hacer tareas para las que necesitaríamos a una persona, pudiendo hacerlo mejor, más rápido y sin las problemáticas que pueda acarrear depender de esta. Pero no es oro todo lo que reluce.

Para obtener grandes resultados hace falta realizar una gran inversión en recursos. Las limitaciones de cómputo y la inexperiencia han sido muy relevantes en todas las dificultades del análisis y desarrollo del sistema. Se necesitan sistemas que provean con la mayor capacidad de cómputo disponible al igual que a profesionales con una gran base teórica y bagaje en estos campos para realizar un buen diseño del sistema, que podrían haber supuesto un gran ahorro de tiempo en etapas futuras del desarrollo, dándole a este proyecto por ejemplo la posibilidad de experimentar con más algoritmos, con mapas distintos, pruebas de cambios en Hardware o la arquitectura del aprendizaje reforzado, y en general, un resultado más satisfactorio.

Además, no siempre es necesario optar por la solución del aprendizaje automático. Existen problemas bien documentados y con soluciones robustas a partir de algoritmos clásicos. Boston Dynamics [15] es una empresa de robótica que aparentemente no hace uso del aprendizaje automático y tiene sistemas capaces de hacer Parkour y sistemas cuadrúpedos capaces de mantener gran estabilidad en entornos dinámicos y peligrosos. No hay que perder la perspectiva de que muchas de las tecnologías no son más que herramientas, y aunque haya una herramienta que resuelva un problema, no significa que no existan otras que también lo puedan hacer.

Aún con todo, no dudamos de que la línea de trabajo a seguir es la que se ha tomado en este Trabajo de Fin de Grado. Los vehículos autónomos son la aplicación más extendida de sistemas similares al propuesto en este TFG, como pueden ser los de la ya mencionada Tesla [16], que a

pesar de encontrar ciertas dificultades en sus planes de mercantilización debido a sobreestimaciones de la tecnología, debates éticos[17] e incidentes aislados[18] empiezan a estar muy presentes en la sociedad. Estos sistemas sin embargo suelen disponer de un número mucho mayor de sensores, tanto en la cantidad como en el tipo de ellos (sonars, cámaras, lidar), que pueden suponer un coste computacional mayor, pero también una representación mucho más precisa para el robot del mundo, lo cuál puede mejorar considerablemente la toma de decisiones.

La línea futura de trabajo que parece natural y lógica para esta investigación es la experimentación con nuevos tipos de sensores, como puedan ser cámaras, y el uso de otros algoritmos de aprendizaje automático que puedan dar una mejor solución ya sea en tiempo de entrenamiento, en resultados o ambos.

Otro aspecto a considerar es que en cualquier caso el mundo es complejo, y para poder enseñárselo a una máquina siempre necesitaremos un gran número de sensores y uno mayor incluso de datos. Esto nos da a entender como se sugirió en el apartado 4 de este TFG que de cara a la escalabilidad y en general eficiencia a la hora de trabajar, el uso de sistemas distribuidos como pueden ser la nube de Azure[19] son la solución más adecuada para lidiar con estos aspectos. Además, si tenemos en cuenta el precio estimado del Hardware en el apartado 10, es incluso probable que estemos optando por una solución no sólo mejor sino también más barata.

Para finalizar, aunque se hayan encontrado muchas problemáticas durante el trabajo y la parte de resultados no haya sido totalmente satisfactoria, creemos que el diseño y funcionamiento del sistema (fuera de la parte del aprendizaje), las soluciones que se han dado a ciertas cuestiones que se han dado durante el desarrollo y en general la metodología de trabajo han sido muy acertadas. Se han respondido muchas dudas que se tenían al principio del desarrollo, se ha aprendido mucho y ha surgido curiosidad por muchos de los temas transversales presentes. Esperamos que todo el trabajo expuesto pueda ser inspirador y de utilidad para el lector, y poder continuar en el futuro trabajando con la base aquí planteada.

# 9. Summary and conclusions.

The fields of artificial intelligence and robotics are capable of providing very good solutions to a wide range of problems, so they are always tools to consider in software development. At the end of the day they are systems capable of doing tasks for which we would need a person, being able to do it better, faster and without the problems that may arise from depending on it. But all that glitters is not gold.

To get great results requires a large investment in resources. Computational limitations and inexperience have been very relevant in all the difficulties of the analysis and development of the system. Systems that provide the highest computational capacity available are required, as well as professionals with a great theoretical background in these fields to perform a good design of the system, which could have lead to a great saving of time in future stages of development, giving this project for example the possibility to experiment with more algorithms, with different maps, testing changes in hardware or the architecture of reinforced learning, and in general, a more satisfactory result.

In addition, it is not always necessary to choose a machine learning solution. There are well-documented problems with robust solutions based on classical algorithms. Boston Dynamics [15] is a robotics company that apparently does not make use of machine learning and has systems capable of doing Parkour and quadruped systems capable of maintaining great stability in dynamic and dangerous environments. We must not lose sight of the fact that many technologies are nothing more than tools, and even if there is a tool that solves a problem, it does not mean that there are not others that can also do it.

Even so, we think that the line of work to follow is the one that has been taken in this Final Degree Project. Autonomous vehicles are the most widespread application of systems similar to the one proposed in this memory, such as those of the aforementioned Tesla [16], which despite encountering certain difficulties in their commercialization plans due to overestimation of the technology, ethical debates[17] and isolated incidents[18] are beginning to be very present in

society. These systems however usually have a much larger number of sensors, both in quantity and type (sonars, cameras, lidar), which can imply a higher computational cost, but also a much more accurate representation of the world for the robot, which can considerably improve decision making.

The future line of work that seems natural and logical for this research is experimentation with new types of sensors, such as cameras, and the use of other machine learning algorithms that can give a better solution either in training time, results or both.

Another aspect to consider is that in any case the world is complex, and in order to teach it to a machine we will always need a large number of sensors and even more data. This could mean that as suggested in section 4 of this TFG that in terms of scalability and overall efficiency when working, the use of distributed systems such as the Azure cloud[19] are the most appropriate solution to deal with these aspects. Moreover, if we take into account the estimated price of the Hardware in section 10, it is even likely that we would be opting for a solution that is not only better but also cheaper.

To conclude, although many problems have been encountered during the work and the results have not been totally satisfactory, we believe that the design and operation of the system (outside the learning part), the solutions that have been given to certain issues that have arisen during the development and in general the methodology of work have been very successful. Many doubts have been answered that we had at the beginning of the development, we have learned a lot and have been curious about many of the cross-cutting issues present. We hope that all the work presented can be inspiring and useful for the reader, and that we can continue working in the future with the basis presented here.

# 10. Presupuesto

Las tecnologías usadas para el desarrollo del proyecto son gratuitas u open source por lo que no añadirán precio al presupuesto más allá del tiempo de trabajo que puedan añadir al proyecto. El precio de las horas de trabajo se estimará utilizando el salario por hora de un ingeniero informático junior. Para el desarrollo se emplearon alrededor de 325 horas, estimando un salario aproximado de 15€ la hora, daría un total de 4875€.

| Partida | Tiempo estimado dedicado |
|---|---|
| Análisis del problema | 20 horas |
| Diseño de la solución | 20 horas |
| Preparación del entorno (Ubuntu, ROS, Gazebo, Python…) | 15 horas |
| Codificación | 80 horas |
| Entrenamientos del sistema | 100 horas |
| Pruebas y revisiones del entrenamiento y código. | 50 horas |
| Redacción de la memoria | 15 horas |
| Tutorías | 10 horas |
| Tareas de la asignatura (grabación de vídeo, tareas de reuniones de seguimiento, curso de preparación para el TFG...) | 15 horas |
| **Total** | 325 horas |

Tabla 10.1: Desglose de horas de trabajo

A este valor habría que añadirle el precio del Hardware utilizado [Tabla 4.1]. Estos valores podrán fluctuar mucho en base al momento en el que se adquieran, sobretodo en el contexto de publicación de esta memoria en la que el mercado de componentes Hardware, más concretamente con el mercado de tarjetas gráficas, a raíz de la situación de pandemia y del auge del minado de criptomonedas. Aún así se intentará hacer una estimación promedio y realista.

| **Componente** | **Especificación** | **Precio estimado** |
| --- | --- | --- |
| Procesador | AMD Ryzen 5 3600 6-Core Processor | 249,90€ |
| Tarjeta gráfica | GP107 [GeForce GTX 1050 Ti] 4GiB | 249,50€ |
| Placa base | B450 AORUS PRO-CF | 96,99€ |
| Memoria RAM | 16GiB | 62,42€ |
| **Total** | | 658,81€ |

Tabla 10.2: Desglose de precios del Hardware utilizado.

Todo esto nos dará como total un presupuesto estimado para el proyecto de: **5533,81 €**

# 11. Bibliografía


[1]Redacción, "Victor," *ConceptoDefinicion.de*, Aug. 30, 2021. [Online]. Disponible: https://conceptodefinicion.de/robot/

[2]"ROS.org." [Online]. Disponible: https://www.ros.org/

[3]OSRF, "Gazebo." [Online]. Disponible: http://gazebosim.org/

[4]"Ficha técnica Pioneer3dx," 2011. [Online]. Disponible: https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf

[5]"Download Ubuntu Desktop," *Ubuntu*. [Online]. Disponible: https://ubuntu.com/download/desktop

[6]Tutorial instalación ros-melodic. [Online] Disponible: http://wiki.ros.org/melodic/Installation/Ubuntu

[7]Documentación python. [Online] Disponible*:* https://www.python.org/

[8]Documentación launchfiles. [Online] Disponible: http://wiki.ros.org/roslaunch

[9]"Q-learning," *Wikipedia.* [Online] Disponible: https://es.wikipedia.org/wiki/Q-learning

[10]OpenAI, "Gym: A toolkit for developing and comparing reinforcement learning algorithms." [Online] Disponible: https://gym.openai.com/

[11]"openai_ros/TurtleBot2 with openai_ros - ROS Wiki." [Online] Disponible: http://wiki.ros.org/openai_ros/TurtleBot2%20with%20openai_ros

[12]A. Robotics, "Extending the OpenAI gym for robotics," *YouTube*. Aug. 19, 2016, [Online]. Disponible: https://www.youtube.com/watch?v=8hxCBkgp95k.

[13]"CUDA Zone," *NVIDIA Developer*, Jul. 18, 2017. https://developer.nvidia.com/cuda-zone

[14]A. Robotics, "Extending the OpenAI gym for robotics," *YouTube*. Aug. 19, 2016. [Online]. Available: https://youtu.be/8hxCBkgp95k.

[15]"Home," *Boston Dynamics*. https://www.bostondynamics.com

[16]"Coches eléctricos, energía solar y limpia," *Tesla*. [Online] Disponible: https://www.tesla.com/es_es

[17]F. Biondi, "Why we still don't have self-driving cars on the roads in 2021," *The Conversation*, Jun. 16, 2021. https://theconversation.com/why-we-still-dont-have-self-driving-cars-on-the-roads-in-2021-162646

[18]G. García, "El extraño accidente mortal de un Tesla Model S sin conductor y sin Autopilot," *Híbridos y Eléctricos*, Apr. 20, 2021.

https://www.hibridosyelectricos.com/articulo/actualidad/extrano-accidente-mortal-tesla-model-s-conductor-autopilot/20210420120205044410.html

[19]"Servicios de informática en la nube," *Microsoft Azure*. https://azure.microsoft.com/es-es/