TecnoCampus
Escola Superior Politècnica

Centre adscrit a la
upf. Universitat Pompeu Fabra Barcelona

**Grado en ingeniería electrónica industrial y automática**

# NAVEGACIÓN AUTÓMOMA DEL TURTLEBOT3 WAFFLE PI

## Memoria Final

**SERGIO VALLE GALLARDO**

**PONENTE: JOSEP LÒPEZ XARBAU**

**PRIMAVERA 2023**

TecnoCampus
Mataró-Maresme

# Agradecimientos

Quiero agradecer a mi profesor ponente Josep Lòpez por la ayuda y la oportunidad de haber podido desarrollar un proyecto del ámbito de la robótica.

Quiero agradecer a Salvador Alepuz ya que me aconsejó a realizar un proyecto que realmente me entusiasma.

Y, por último, quiero agradecer a mis padres por haberme dado la oportunidad de haber estudiado lo que me gusta.

## Resum

Aquest projecte consisteix en el desenvolupament d'un entorn on TurtleBot3 Waffle Pi ha de ser capaç d'aconseguir realitzar un camí en un entorn desconegut, fent ús de les diferents técniques com el reconeixement de patrons i les xarxes neuronals, identificant diferents classes d'objectes que indiquen quina acció ha de realitzar el TurtleBot en cada moment en temps reial.

## Resumen

Este proyecto consiste en el desarrollo de un entorno donde TurtleBot3 Waffle Pi tiene que ser capaz de conseguir realizar un camino en un entorno desconocido, haciendo uso de las diferentes técnicas como el reconocimiento de patrones y las redes neuronales identificando diferentes clases de objetos que indican que acción debe realizar el TurtleBot3 en cada momento en tiempo real.

## Abstract

This project consists of the development of an environment where TurtleBot3 Waffle Pi has to be able to make a path in an unknown environment, making use of different pattern recognition techniques and neural networks, identifying different clases of objects that indicate that action must be performed by TurtleBot3 at each moment in real time.

# Índice

# Índice de figuras

# Índice de tablas

# Glosario de términos

| | |
|---|---|
| RNA | Red Neuronal Artificial |
| SBC | Single Board Computer |
| ROS | Robot Operating System, Open Robotics |
| MCU | Microcontrolador |
| CSI | Por corriente |
| GNOME | Network Object Model Environment |
| IMU | Unidad de Medida Inercial |
| API | Application Programming Interface |
| DQN | Deep Q Networks |
| SSH | Secure SHell |

# 1.Objetivos

## 1.1 Propósito

El propósito principal de este proyecto es conseguir que el robot denominado TurtleBot3 Waffle Pi sea capaz de desplazarse en un entorno desconocido.

## 1.2 Finalidad

La finalidad del proyecto es realizar un estudio del entorno del TurtleBot3 Waffle Pi con el objetivo de aplicar diferentes técnicas de reconocimiento de patrones y redes neuronales con un procedimiento de aprendizaje de propagación posterior para la navegación autónoma de este robot.

La principal finalidad para la navegación autónoma del TurtleBot3 Waffle Pi es conseguir que el robot sea capaz de realizar un camino en un entorno totalmente desconocido, consiguiendo esto, gracias a como ya se ha comentado anteriormente mediante el reconocimiento de patrones y las redes neuronales identificando diferentes clases que indican que acción debe realizar el robot en cualquier momento y con cualquier obstáculo que se interponga.

## 1.3 Objeto

Lo que se quiere conseguir con este proyecto es la aplicación de diferentes técnicas de reconocimiento de patrones y redes neuronales con el procedimiento de aprendizaje de propagación posterior para la navegación autónoma del robot, en este caso del TurtleBot3 Waffle Pi.

Para conseguir realizar este proyecto, es necesario llevar a cabo un estudio detallado de las especificaciones técnicas del robot a usar, y estudiar los componentes que forman la estructura de este. Más allá de la parte de hardware, también se hace un estudio del software a utilizar en este proyecto, para así tener una familiarización con dicho software y entender su funcionamiento a la perfección.

## 1.4 Alcance

En este proyecto se realiza un estudio de la robótica y de las redes neuronales, y qué posibilidades puede ofrecer, específicamente de cara al TurtleBot3.

Para el manejo y el desarrollo de funciones propuestas para el TurtleBot3, como el reconocimiento de patrones y las redes neuronales, se ha realizado un estudio de como implementar estas funciones de cara al robot deseado.

Respecto el uso del TurtleBot3 Waffle Pi se ha realizado un estudio del sistema operativo ROS, para así obtener una familiarización con dicho entorno y entenderlo a la perfección. Se realiza una programación especifica con el lenguaje Python para que el robot sea capaz de navegar en un entorno desconocido y saber como debe reaccionar ante imprevistos u obstáculos.

Se estudia la navegación en el entorno desconocido con simulaciones mediante el PC, haciendo uso de herramientas como Gazebo donde se trabaja en un entorno 3D, o como RVIZ para realizar mapeados. Además, se realizan todos los pasos pertinentes para que todo aquello que se ha logrado alcanzar con la simulación, se pueda realizar con el robot en un entorno real.

Para acabar, este proyecto contiene una planificación, viabilidad técnica, viabilidad medioambiental, una viabilidad económica, y unos objetivos y especificaciones técnicas, donde se explica detalladamente que es lo que se ha realizado durante la realización de este proyecto.

# 2. Revisión de antecedentes y necesidades de información

## 2.1 Reconocimiento de patrones y redes neuronales

Se puede entender como reconocimiento de patrones aquella actividad de identificar posibles objetos o diferentes comportamientos que suelen ser recurrentes y poder clasificarlos. Para definirlo de una forma más exacta, el reconocimiento de patrones es una disciplina que se encarga de identificar figuras, reconocer formas o leer patrones, con el objetivo de recoger toda esta información almacenada sobre aquello que se está estudiando y asignarlo a una clase [1].

A continuación, se va a explicar cómo funciona el reconocimiento de patrones dividiéndolos en cuatro sencillos apartados.

**Adquisición de datos**: la adquisición de datos se consigue mediante el sensor, el cual es el dispositivo encargado de realizar esta tarea.

**Extracción de característica:** es el proceso en el que se generan diferentes características que pueden ser usadas en el proceso de clasificación de estos datos recopilados.

**Selección de atributos:** es la selección de atributos más adecuada para describir a los objetos.

**Clasificación de objetos:**

- **Supervisada:** la clasificación supervisada conocida también como clasificación con aprendizaje, se basa en la disponibilidad de áreas de entrenamiento. Este tipo de clasificación consiste básicamente en que, las áreas que se conocen desde un principio la clase a la cual pertenecen, y que servirán para generar una signatura espectral específica de cada una de las clases [1].

  A continuación, se explican tres métodos de la clasificación supervisada:

  - **Vecino más cercano:** se introduce un nuevo objeto en aquella clase donde esté el objeto de la muestra original que más se le parece [1].

  - **Funciones discriminantes:** si existen múltiples clases se busca un conjunto de funciones gi, y el nuevo objeto se tiene que ubicar en aquella clase específica donde la función escoja el valor más grande [1].

  - **Redes neuronales artificiales:** las redes neuronales artificiales también conocidas y representadas como RNA, se supone que imitan a las redes neuronales reales en el desarrollo de las tareas de aprendizaje que se puedan llegar a realizar [1].

- **Parcialmente supervisada:** la clasificación parcialmente supervisada también conocida como aprendizaje parcial. En este tipo de clasificación, existe una única muestra de diferentes objetos en alguna de las clases definidas [1].

- **No supervisada:** la clasificación no supervisada también conocida como clasificación sin aprendizaje, donde se utilizan algoritmos de clasificación automática multivariante que se van agrupando, formando clases todos aquellos individuos más próximos [1].

  - **Restringida:** el número de clases en las que se formará la muestra está definido ya previamente.

  - **Libre:** el número de clases en las que se formará la muestra depende únicamente de los datos.

  Algunos métodos de la clasificación no supervisada son el Simple Link, ISODATA y C-Means.

Una vez ya se ha comentado profundamente que es el reconocimiento de patrones, como está distribuido y los diferentes tipos de clasificación, se puede iniciar la búsqueda de información sobre las redes neuronales, también ya comentada en un método de la clasificación supervisada.

Las redes neuronales trabajan sobre la base del reconocimiento de patrones y estas redes pueden obtener conocimiento a partir de diferentes ejemplos. La característica más destacada de las redes neuronales es la forma de adquirir dicho conocimiento, ya que no hace uso de una programación directa, sino que adquiere este conocimiento en base a ejemplos ya realizados, todo esto, gracias a un algoritmo de aprendizaje.

Este tipo de sistemas como son las redes neuronales artificiales, son capaces de aprender y se forman a ellos mismos. Suelen darse a reconocer en áreas donde la detección de soluciones es difícil de conseguir con la programación que es convencional.

El objetivo principal es resolver todos aquellos problemas de la misma forma que lo haría el cerebro humano.

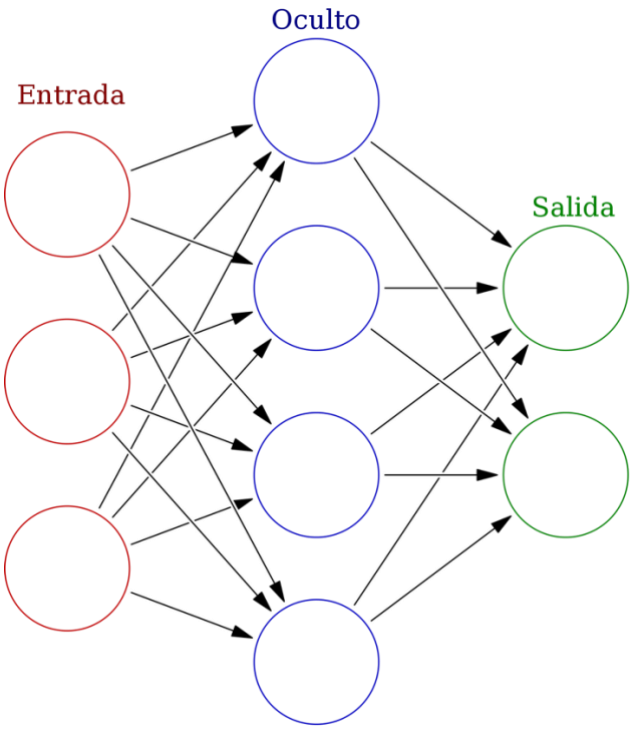


Figura 2. 1 Esquema redes neuronales artificiales

Fuente: [2]

En la anterior figura se puede observar el esquema principal de cómo actúan las redes neuronales. Cada nodo circular representa a una neurona artificial, y cada flecha que provoca la conexión entre dichos nodos, representan una conexión desde la salida de una neurona a la entrada de otra.

## 2.2 TurtleBot3

TurtleBot3 es una nueva generación de robots móviles que están destinados al ámbito del desarrollo de productos, de la investigación y de la educación, todo esto haciendo uso del entorno ROS [3].

El objetivo principal del TurtleBot3 es reducir de una forma drástica el tamaño y bajar el precio sin hacer el sacrificio de rebajar la capacidad, la funcionalidad y la calidad.

Es un robot modular, el cual dispone de piezas opcionales como chasis, computadoras y diferentes sensores. Como ya se ha comentado anteriormente, el TurtleBot3 se puede personalizar de diferentes maneras, aplicando últimos avances técnicos del SBC, el sensor de profundidad y la tecnología de impresión 3D [4].

Actualmente en el mercado existen dos tipos de TurtleBot3, los cuales se verán a continuación.

No obstante, para la realización de este proyecto se va a trabajar con el TurtleBot3 Waffle Pi.

### 2.2.1 TurtleBot3 Burger

En esta variante del TurtleBot3, primero, se explican las diferentes especificaciones técnicas y características que este robot puede proporcionar, y luego se explican los componentes que contiene.

#### 2.2.1.1 Especificaciones técnicas i características

A continuación, se observan las especificaciones técnicas correspondientes al TurtleBot3 Burger [4].

- Máxima velocidad de translación: 0,22 m/s
- Máxima velocidad de rotación: 162,72 grados/s
- Máxima carga útil: 15 kg
- Dimensiones: 138 x 178 x 192 mm

- Peso (SBC + batería + sensores): 995 g
- Autonomía: 2h y 30 min
- Tiempo de carga: 2h y 30 min
- Conexión para el ordenador: USB
- IMU: 3 ejes giroscopio, 3 ejes acelerómetro y 3 ejes magnetómetro.
- 3.3 V / 800 mA
- 5 V / 2ª
- 12 V / 1 A
- Pines de expansión: 18 pines GPIO, 32 pines Arduino
- Dynamixed puertos
- Diferentes secuencias de sonidos programables
- Arduino LED x 1
- 2 botones
- Batería del polímero de litio 11.1 V 1800 mAh / 19.98 Wh 5C
- Adaptador de recarga:
    - Input: 100-240V, AC 50/60 Hz, 1.5 A máx.
    - Output: 12 Vdc, 5 A

Las características se pueden apreciar en la siguiente figura:

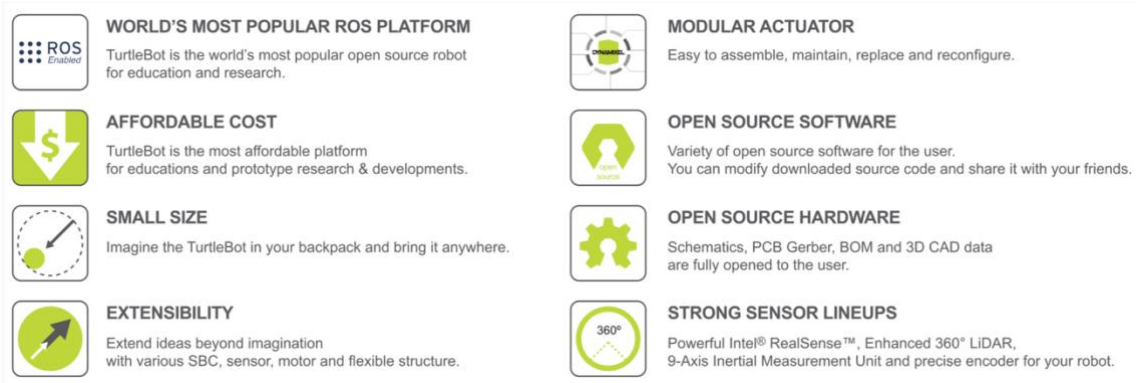


Figura 2. 2 Características del TurtleBot3 Burger

Fuente: [4]

### 2.2.1.2 Componentes

El cuerpo del robot queda distribuido en cuatro niveles diferentes. Como ya se ha comentado anteriormente sus dimensiones son de 138 mm x 178 mm x 192 mm.

Figura 2. 3 Estructura TurtleBot3 Burger

Fuente: [5]

El primer nivel del robot de estos cuatro comentados, empezando desde la base de este, contiene la batería y los motores DYNAMIXEL XL430, y como se observa en la Figura 2.3, también es

para las dos ruedas que contiene este robot; en el segundo nivel se encuentra la tarjeta OpenCr; en el tercer nivel tiene situada la Raspberry PI 3, y en el último de estos niveles se encuentra el sensor de distancia láser LDS-01.

**Motor DYNAMIXEL (XL430-W250-T):**

El DYNAMIXEL X es una nueva serie de actuadores de alto rendimiento en red.

En este caso se utiliza el DYNAMIXEL XL, este actuador adopta nuevas funcionalidades, permitiendo el modo de control de 360 grados con su *encoder* magnético sin contacto y esa estructura de conjunto de caja trasera hueca [6].


Figura 2. 4 Motor DYNAMIXEL (XL430-W250-T)

Fuente: [7]

A continuación, se explican diferentes características que este tipo de motor puede llegar a ofrecer.

- Control de velocidad
- Control de posición
- Control extendido
- Control PWM
- Resolución: 4096 pulso/rev
- Voltaje de entrada: entre 6,5 V y 12 V

**Placa OpenCR 1.0:**

La placa OpenCR 1.0 es un controlador de robot de código abierto integrado con una potente MCU de la línea ARM Cortex-M7[8].

A continuación, se verán diferentes características que puede llegar a proporcionar la placa OpenCr 1.0:

- Controlador principal en el entorno ROS, es accesible y está abierto al público. Tiene compatibilidad con RS-485 y TTL para poder controlar los actuadores Dynamixel.
- Ofrece UART, CAN, JTAG, I2C y otros elementos de comunicación.
- También ofrece herramientas de desarrollo como es el Arduino IDE.
- Proporciona pines digitales y analógicos de entrada/salida que pueden actuar con diferentes sensores.

En la siguiente figura se puede observar cómo está la placa OpenCR 1.0 distribuida y todas sus propiedades.

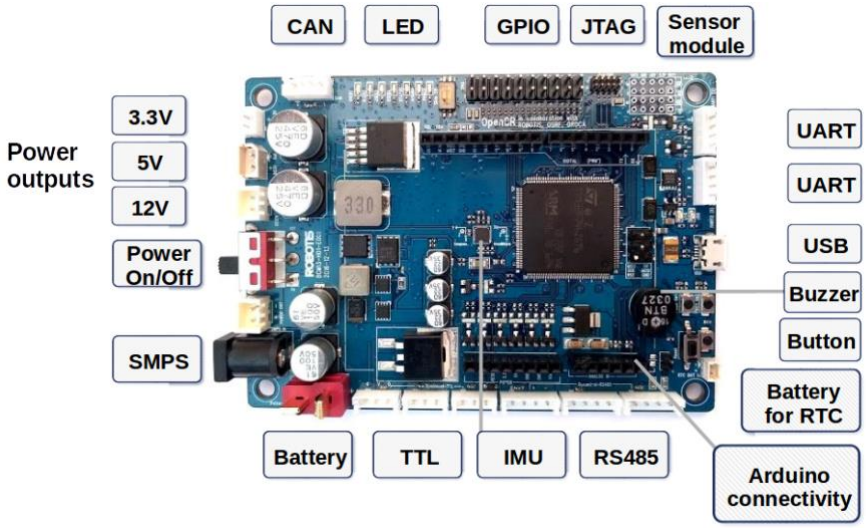


Figura 2. 5 OpenCR 1.0

Fuente: [8]

**Raspberry Pi 3:**

La Raspberry Pi 3 Model B es una nueva versión que cuenta con la incorporación de las siguientes características:

- CPU Cortex-A53 de cuatro núcleos, de 64 bit y 1.2 GHz, lo que provoca que la velocidad respecto a su versión anterior mejore en un 50% [10].
- Conexión Wi-Fi 802.11
- Bluetooth 4.0
- 4 puertos USB 2.0 (USB x2, HDMI, Puerto de cámara CSI, puerto microSD)
- 40 pines GPIO

Figura 2. 6 Raspberry Pi 3 Model B

Fuente: [9]

**Sensor de distancia láser LDS-01:**

El LDS-01 es un sensor de distancia láser, el cual es capaz de recopilar un seguido de datos gracias a la habilidad que tiene de girar 360 grados [11].

Las principales características que ofrece este sensor son las siguientes:

- **Voltaje de suministro:** 5V
- **Consumo de corriente:** 400 mA aproximadamente

- **Corriente rápida:** 1A
- **Distancia de detección:** entre 120mm i 3.500mm
- **Tasa de muestreo:** 1,8 KHz
- **Gama angular:** 360 grados
- **Resistencia a luz ambiental:** 10.000 lux

Una vez vistas estas características, el sensor se conecta a la Raspberry Pi 3 mediante el uso de un USB.

## 2.2.2 TurtleBot3 Waffle Pi

En esta variante del TurtleBot3 primero se verán explicadas las diferentes especificaciones técnicas y características que este robot puede proporcionar, y luego se explicarán los componentes que contiene.

### 2.2.2.1 Especificaciones técnicas i características

A continuación, se observan las especificaciones técnicas correspondientes al TurtleBot3 Waffle Pi [12].

- Máxima velocidad de translación: 0,26 m/
- Máxima velocidad de rotación: 104,27 grados/s
- Máxima carga útil: 30 kg
- Dimensiones: 281 x 306 x 141 mm
- Peso (SBC + batería + sensores): 1,8 kg
- Autonomía: 2h
- Tiempo de carga: 2h y 30 min
- Dynamixel XM430-W210-T

- Raspberry Pi
- OpenCR 1.0
- Raspberry Pi cámara
- Sensor 360 grados LDS

Las características se pueden apreciar en la siguiente figura:


Figura 2. 7 Características del TurtleBot3 Waffle Pi

Fuente: [12]

### 2.2.2.2 Componentes

El TurtleBot3 Waffle Pi tiene exactamente los mismos componentes que el TurtleBot3 Burger, estos componentes son la placa OpenCR1.0, la Raspberry Pi 3 y el sensor láser LDS-01, como ya se han explicado en el anterior apartado detenidamente, no se cree necesario volver a explicar todos estos componentes. Además de los componentes que comparten, el TurtleBot3 Waffle Pi incorpora un nuevo motor diferente al del robot anterior, en este caso, se trata del motor Dynamixel XM430-W210-T, y también incorpora otro nuevo componente como es la Raspberry Pi cámara.

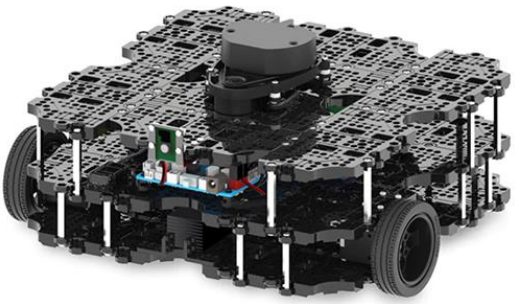

Figura 2. 8 Estructura TurtleBot3 Waffle Pi

Fuente: [13]

**Motor DYNAMIXEL (XM430-W210-T):**

El DYNAMIXEL X es una nueva serie de actuadores de alto rendimiento en red.

En este caso, se trabaja con el motor DYNAMIXEL XM, el cual tiene la misma estructura mecánica que la serie XL, pero con unas características diferentes [14].


Figura 2. 9 Motor DYNAMIXEL (XM430-W210-T)

Fuente: [15]

A continuación, se comentan diferentes características que este tipo de motor puede llegar a proporcionar.

- Control de corriente
- Control de velocidad
- Control de posición
- Control extendido
- Control de posición basado en la corriente
- Control PWM
- Resolución: 4.096 pasos
- Voltaje de entrada: entre 10 V y 14,8 V
- Corriente en espera: 40mA

**Raspberry Pi Cámara:**

Módulo de cámara de alta definición compatible con todos aquellos modelos de Raspberry Pi. Proporciona captura de imágenes de alta sensibilidad, baja diafonía y bajo ruido en un ultra diseño pequeño y ligero. [16]

El módulo de la cámara se conecta a la Raspberry Pi a partir del conector CSI, diseñado específicamente para conectarse a cámaras. El bus CSI es capaz de alcanzar velocidades de datos extremadamente altas y transporta exclusivamente datos de pixeles al procesador. [16]

A continuación, se observan un seguido de características sobre este componente.

- **Sensor de imagen:** Sony IMX 219 PQ CMOS
- **Resolución:** 8-megapixel
- **Resolución de imagen fija:** 3280 x 2464
- **Velocidad máxima de transferencia de imágenes:** 1080 p;30 fps
- **Dimensiones:** 23,86 x 25 x 9mm

- **Peso:** 3g

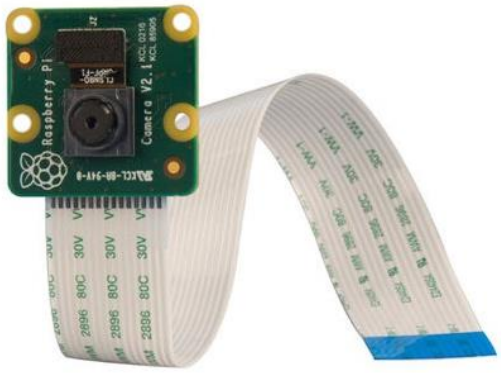


Figura 2. 10 Raspberry Pi cámara

Fuente: [17]

## 2.3 Manera de actuar cuando hay obstáculos

En este apartado, se presentan diferentes alternativas para que en el caso de que aparezca un obstáculo móvil, el robot mediante el reconocimiento de patrones pueda identificar o actuar de una forma u otra dependiendo del obstáculo que tenga enfrente. Cuando se habla de obstáculo en este apartado, se refiere únicamente a un obstáculo móvil, cuando se trate de un obstáculo fijo se comentará directamente.

A continuación, se contemplan varias posibilidades a las que el robot puede optar:

- **Girar y buscar una alternativa:**

  Cuando el robot identifica un obstáculo enfrente suyo, este ha de ser capaz de, si tiene el suficiente espacio para hacerlo, girar y buscar un camino alternativo al propuesto inicialmente. Esta respuesta del robot requiere de una nueva ruta, objetivo que para el robot puede ser más complejo, pero no ha de tener ningún problema en realizarlo.

- **Esperar a que el obstáculo desaparezca:**

En esta segunda alternativa, el robot identifica el obstáculo enfrente suyo. Una vez identificado el obstáculo, en vez de buscar una nueva ruta como en el anterior caso, el robot esperará el tiempo necesario hasta que dicho obstáculo desaparezca de enfrente suyo y pueda continuar su ruta sin ningún tipo de inconveniente.

# 3. Objetivos y especificaciones técnicas

En este apartado se trata de definir todos aquellos objetivos que se pretenden conseguir con la realización del proyecto.

Para establecer estos objetivos, primero se ha de recolectar la suficiente información sobre las diferentes características que el TurtleBot3 puede proporcionar, tanto de software como hardware que puede llegar a aportar.

Como ya se ha visto en el apartado 2.2, se ha realizado una investigación profunda y un análisis detallado sobre estos aspectos comentados en el párrafo anterior.

A continuación, se establecen los objetivos a conseguir:

- Estudiar e investigar el sistema operativo denominado ROS.
- De forma virtual, conseguir que el TurtleBot3 sea capaz de moverse por un entorno desconocido haciendo uso de redes neuronales.
- Poner en marcha el TurtleBot3, en este caso, el TurtleBot 3 Waffle Pi.
- Mover el robot real por un camino en un entorno totalmente desconocido.
- Establecer redes neuronales para hacer frente a los diferentes obstáculos que surjan en un entorno real.

Estos cuatro objetivos son aquellos con los que se trabajará durante el desarrollo del proyecto. Al inicio, se realizará un ensayo virtual para ver si realmente se han obtenido los objetivos a cumplir, posteriormente, se realizará un ensayo real con el robot que se dispone en el laboratorio.

Una vez se han establecido y aclarado los objetivos definitivos, a continuación, se detallan las especificaciones técnicas correspondientes a los objetivos comentados anteriormente.

- **Estudiar e investigar el sistema operativo denominado ROS:** hacer una investigación profunda de un sistema operativo como es en este caso ROS, así conociendo herramientas como Gazebo, RViz, y así lograr tener visualizaciones 3D y realizar mapeados SLAM.

- **De forma virtual, conseguir que el TurtleBot3 sea capaz de moverse por un entorno desconocido haciendo uso de redes neuronales:** a partir de diferentes programas de mapeado que proporciona ROS, del entorno 3D Gazebo, y a través de la programación, conseguir que el robot sea capaz de forma autónoma de moverse por entornos desconocidos de forma virtual mediante redes neuronales, para posteriormente, realizar el mismo proceso, pero de forma real.
- **Poner en marcha el TurtlelBot3:** se ha de configurar e instalar todos aquellos parámetros necesarios del robot y del ordenador pertinentes, para poder trabajar a la perfección con el robot en un entorno real.
- **Mover el robot real por un camino en un entorno totalmente desconocido:** realizar un código mediante Python para que el robot de forma autónoma pueda desplazarse en cualquier entorno deseado.
- **Establecer redes neuronales para hacer frente a los diferentes obstáculos que surjan en un entorno real:** la detección de todos los obstáculos del entorno real gracias al trabajo de las redes neuronales y del sensor LDS para evitar colisiones, después de haber realizado un período de entrenamiento.

# 4. Generación y planteamiento de alternativas

En este apartado se plantean diferentes alternativas para poder realizar el proyecto, pero como ya se ha comentado anteriormente, trabajar con TurtleBot3 está pensado básicamente para ser programado con el entorno ROS.

Por esta razón, no se plantea ninguna otra alternativa que no sea la de programar con el entorno ROS. El hecho de trabajar con ROS implica que se puedan encontrar diferentes formas de programar los diferentes componentes del TurtleBot3.

Actualmente se encuentran muchas distribuciones dentro del entorno ROS. El propósito de las distribuciones es dejar que los desarrolladores funcionen contra una base de código relativamente estable hasta que estén preparados para hacer todo lo posible.

La distribución más reciente se denomina ROS Noetic Ninjemys, la cual es la última versión que ha lanzado ROS al mercado, otra distribución que está en su periodo de vida es el ROS Melodic Morenia, la cual fue lanzada el 23 de mayo del 2018. A parte de estas distribuciones más recientes, se va a comentar una más antigua como es el ROS Kinetic Kame, la cual dejó de tener soporte y mantenimiento por la compañía en el año 2018.

Ya planteadas las diferentes posibilidades, se ha de hacer un estudio para escoger aquella distribución definitiva con la cual se va a trabajar finalmente.

# 5. Selección de la alternativa óptima

Una vez se han planteado las diferentes alternativas que puede haber en el apartado anterior, se ha de escoger una de estas soluciones.

El ROS Noetic Ninjemys, todo y ser la distribución más reciente que se ha sacado al mercado, no se ha encontrado la más adecuada para trabajar en este proyecto con el TurtleBot3.

Después de una búsqueda de información exhaustiva sobre la selección de aquella alternativa que puede ser más adecuada para trabajar con el entorno ROS y el TurtleBot3, en este caso, el TurtleBot3 Waffle Pi. Se ha llegado a la conclusión, que el entorno ROS Kinetic Kame es la mejor opción para trabajar con ello.

Esta elección, es debido a que esta distribución proporciona muchas librerías relacionadas con TurtleBot3, este hecho facilita los recursos a los cuáles se puede acceder, dando lugar, a que sea la manera más asequible para lograr el objetivo de este proyecto debido al tiempo disponible que se tiene para realizarlo.

ROS Kinetic Kame dispone de muchas guías y proporciona diferentes librerías, con las cuales se pueden realizar mapeados por el entorno de donde se encuentra el robot, y así, poder seguirlo y ver su comportamiento en tiempo real desde el ordenador en el que se esté trabajando, sin tener la necesidad de seguir con la vista al robot real.

Después de varias pruebas y de varios errores tras intentos de instalación, se ha descartado trabajar con la versión ROS Kinetic, estos diferentes errores han podido ser debido a que este tipo de versión ya no recibe soporte. Por esta misma razón, se ha decidido trabajar finalmente con la versión ROS Melodic.

ROS Melodic es una versión más reciente respecto a la del ROS Kinetic, la cual, si que sigue recibiendo soporte, y, además, dispone de bastantes guías y librerías para poder trabajar de una manera más sencilla con el objetivo deseado, que es realizar mapeados de forma autónoma mediante el reconocimiento de patrones y las redes neuronales.

# 6. Marco conceptual

## 6.1 Ubuntu

Ubuntu también conocido como un tipo de *distro,* es un sistema operativo gratuito que se basa en el núcleo de Linux. Ubuntu es de código abierto, esto quiero decir que el usuario o desarrollador pueda manipular su código, crear e instalar toda la cantidad de copias que se desee. [18]

Utiliza entornos de escritorio Linux para su interfaz, el conocido como GNOME es el predeterminado. A continuación, se observará como se presenta este entorno de escritorio con la siguiente figura [18].

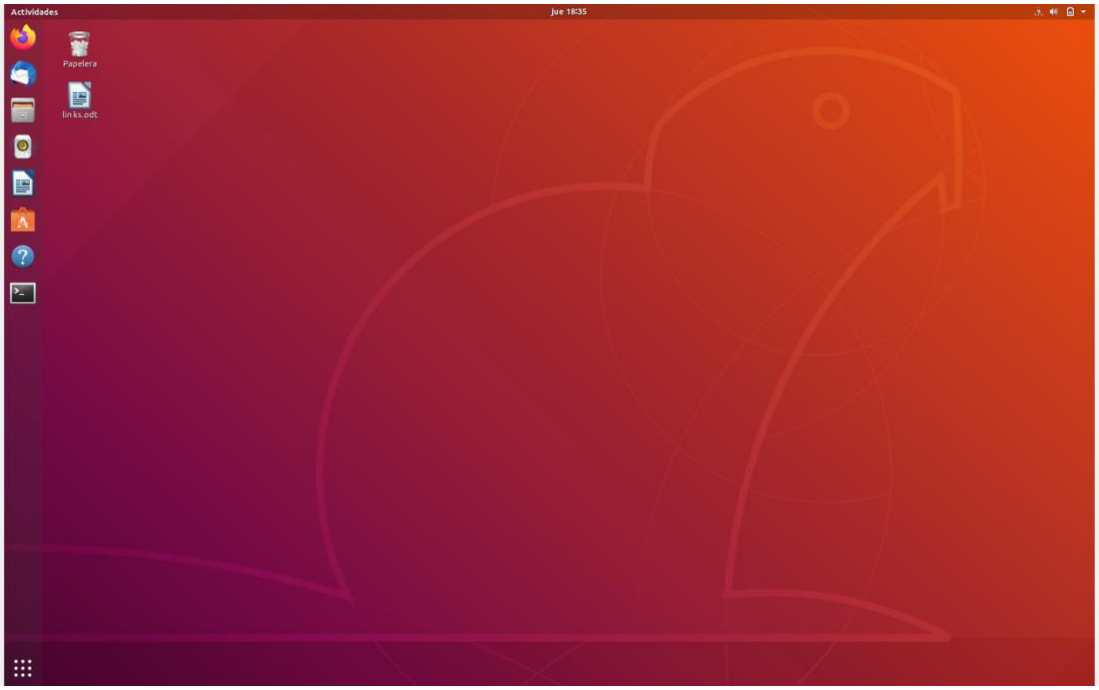


Figura 6. 1 Escritorio Linux

Fuente: Elaboración propia

Como ya se ha comentado anteriormente Ubuntu es de código abierto, pasa por diferentes revisiones para tener controlada su seguridad. Este sistema emplea AppArmor, AppArmor es un módulo de seguridad de Linux, el cual puede reducir considerablemente las posibles violaciones de seguridad [23].

## 6.2 ROS

Primero se ha de saber y conocer con qué entorno se va a trabajar. Para ello, se va a realizar una búsqueda profunda de información sobre que es realmente el entorno ROS y porqué se utiliza, y los diferentes conceptos principales que contiene.

ROS es una estructura de programación libre que permite el desarrollo de programas para robots, proporcionando funciones parecidas a un sistema operativo. ROS implementa los servicios típicos de un sistema operativo como el desarrollo de hardware, es decir, controladores de dispositivos a bajo nivel.

El entorno ROS (Robot Operating system) proporciona las librerías y herramientas necesarias de software para ayudar al usuario a crear diferentes aplicaciones robóticas.

Por lo tanto, ROS dispone de una gran cantidad de librerías de otros ejemplos ya realizados y comprobados. Esto facilita la tarea de programación del robot, ya que debido a que haya otros ejemplos, se pueden coger algunas estructuras de estos programas y trabajar sobre dichos programas.

### 6.2.1 Tipos de software

ROS contiene diferentes tipos de software, exactamente estos tipos son con exactitud cuatro. Por un lado, tiene las distribuciones, por otro lado, los paquetes, también contiene las librerías del núcleo ya comentadas, y, por último, las herramientas comunes.

**Distribuciones:**

Una distribución ROS es un conjunto versionado de paquetes ROS. El propósito de estas distribuciones que nos proporciona ROS es permitir a los diferentes desarrolladores trabajar contra una base de código estable hasta que estén listos para avanzar todo [19].

Hay muchos tipos de robots que llegan a utilizar el entorno ROS, los cuales tienen diferentes necesidades, así que, cada comunidad puede preparar y desarrollar sus propias distribuciones.

**Paquetes**

Los paquetes ROS son creados por desarrolladores, una vez se tienen estos paquetes finalizados se puede observar qué proporcionan. Estos paquetes contienen diferentes funciones y drivers definidos, están disponibles para todo aquel usuario que desee hacer uso de ellos a través de la comunidad [22].

Los paquetes siguen un tipo de esquema denominado esquema modular, esto permite emplear estos paquetes en muchos tipos de aplicaciones diferentes.

**Librerías de núcleo**

ROS proporciona diferentes librerías cliente, aunque las principales librerías compatibles son con el lenguaje de programación C++ y Python. La librería para el lenguaje C++ conocida como *roscpp*, y la librería para el lenguaje Python conocida como *rospy* [19].

También se dispone de diferentes pilas y paquetes ROS que proporcionan funcionalidad de nivel superior. El soporte para dicha funcionalidad en varios lenguajes se puede ver a continuación.

- ROS: funcionalidad de nivel superior disponible a partir de temas y servicios ROS.
- C++: funcionalidad de nivel superior disponible en las bibliotecas C++.
- Python: funcionalidad disponible en módulos y paquetes de Python.

**Herramientas:**

El entorno ROS puede llegar a proporcionar una gran cantidad de herramientas con las cuales el usuario puede interactuar.

En este apartado se comenta únicamente una herramienta, debido a la gran cantidad de herramientas que ROS nos puede llegar a proporcionar.

**Gazebo:**

Gazebo es una herramienta donde se pueden realizar diferentes simulaciones de forma realista, tanto de iteraciones con objetos como la dinámica y los sensores de robots, como es el caso del TurtleBot3. Gazebo funciona mediante el entorno ROS, lo cual permite simular el comportamiento del robot al utilizar varios paquetes desarrollados por ROS [24].


Figura 6. 2 Herramienta Gazebo

Fuente: [25]

**RVIZ:**

RVIZ es una herramienta de simulación y visualización 3D, que permite combinar diferentes datos de sensores, modelos de robots y otros datos 3D en una vista combinada entre ellos.

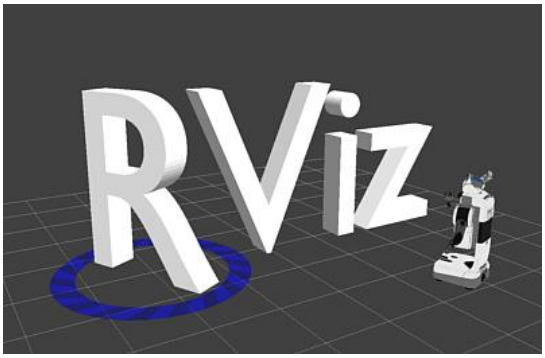


*Figura 6. 3 Herramienta RViz*

Fuente: [20]

### 6.2.2 Conceptos principales de ROS

A continuación, se pueden observar los diferentes conceptos principales del entorno ROS:

- **Nodos:** los nodos son los procesos que realizan la computación. Un sistema de control de robots está compuesto por muchos nodos. A continuación, se explica un ejemplo para acabar de entender este concepto, un nodo controla un láser, un nodo controla los diferentes motores de las ruedas, un nodo se encarga de la localización, un nodo realiza una trayectoria de planificación, y así sucesivamente. Un nodo ROS se escribe con el uso de una librería como roscpp o rospy [21].

  ROS tiene diferentes herramientas para gestionar los nodos y proporcionar información sobre ellos como rosnode. La herramienta rosnode es una herramienta de línea de comando para mostrar información sobre los nodos. A continuación, se observan varias de estas posibilidades [21]:

  - **rosnode info node:** muestra información sobre el nodo.
  - **rosnode kill node:** elimina el nodo o envía un mensaje para eliminar el nodo.
  - **rosnode ping node:** muestra la conectividad con el nodo.
  - **rosnode cleanup:** limpia la información de registro para nodos inaccesibles.
  - **rosnode lis:** muestra una lista con los nodos activos.
- **Master:** el ROS Master proporciona el registro de nombre y busca el gráfico de computación final. Sin el maestro, los nodos no serían capaces de encontrar los otros nodos o intercambiar mensajes [21].
- **Servidor de parámetros:** permite almacenar datos por la clave en una localización central [21].
- **Mensajes:** los nodos se comunican a partir del paso de los mensajes. Un mensaje es una estructura de datos compuestos. Un mensaje está compuesto por una combinación entre tipo primitivo y mensajes. Los tipos primitivos de datos (punto flotante, booleano, entero, ...) [21].

- **Temas:** son canales de comunicación para transmitir datos. Pueden transmitir sin una comunicación directa entre nodos. Un tema puede tener varios suscriptores [21].

  Cada tema está tipado por un tipo de mensaje de ROS que se publica, los nodos pueden recibir mensajes de un tipo determinado. Un nodo puede suscribirse a un único tema si tiene el mismo tipo de mensaje [21].

- **Servicios:** el modelo de publicación / suscripción es un paradigma de comunicación muy flexible, pero en muchos casos el transporte en un único sentido no es suficiente para las iteraciones de petición y respuesta que normalmente se requieren en un sistema distribuido. La petición y respuesta se realiza a través de los servicios, que se definen a partir de una estructura de mensajes: una para la petición y otra para la respuesta. Un nodo proporciona un servicio con un nombre y un cliente utiliza este servicio proporcionando, haciendo uso del envío del mensaje de petición y espera a la respuesta. La librería del cliente ROS generalmente presenta esta interacción como si fuera una llamada a un procedimiento remoto [21].

- **Bolsas:** las bolsas son un formato para guardar y reproducir nuevos mensajes de ROS. Son un mecanismo importante para el almacenaje de datos, como puede ser la lectura de sensores, que pueden ser difícilmente adquiridos, pero son necesarios para poder desarrollar y programar diferentes algoritmos [21].

## 6.3 TensorFlow

TensorFlow es una biblioteca de código abierto con la finalidad de utilizarse para Machine Learning, su uso es para el desarrollo del aprendizaje automático y de la inteligencia artificial a través de un rango de tareas [27].

Es una librería que fue desarrollada por Google para cumplir y satisfacer todas aquellas necesidades a partir de las redes neuronales artificiales.

TensorFlow permite construir y entrenar redes neuronales con el objetico de la detección de diferentes patrones, obstáculos o diversos razonamientos usados por los humanos.

A continuación, se observan diferentes ejemplos que se ven hoy en día en diferentes aspectos del uso del TensorFlow:

- Reconocimiento de imagen: se pueden encontrar diferentes capas de entrada y salida. Las redes neuronales aprenden de forma automática representaciones abstractas de los datos que reciben de imágenes. Las neuronas detectan líneas, texturas, formas, para reconocer una imagen [27].
- Análisis de sentimiento: llegar a predecir los gustos y las necesidades de los clientes, ayuda a la gestión de la relación con los usuarios [27].
- Diagnósticos médicos: se analiza las radiografías a partir de la inteligencia artificial, ya que las redes neuronales pueden llegar a encontrar anomalías en estos procesos médicos [27].

Tras ver varios ejemplos con los que se puede trabajar con TensorFlow, en este proyecto se centra y se utiliza para lograr la finalidad de este trabajo.

## 6.4 Keras

Keras es una biblioteca de código abierto que se ejecuta sobre frameworks como el mencionado anteriormente el TensorFlow. Es una API de redes neuronales escrita en lenguaje Python, diseñada para ser modular, fácil de usar y con un manejo rápido sobre ésta [28].

A continuación, se pueden observar un seguido de características sobre esta biblioteca denominada Keras [28]:

- Está diseñada para ser fácil de ampliar, modular e intuitiva.
- Proporciona la posibilidad de combinar diferentes módulos de capas neuronales, esquemas de inicialización, optimizadores para crear nuevos módulos.
- Facilidad para añadir nuevos módulos, como nuevas clases y funciones.

Keras puede utilizarse para predecir la clasificación de imágenes, extraer características y configurar modelos sobre un conjunto de clases diferentes. Los modelos Keras se pueden desplegar en una gran cantidad de plataformas [28].

# 7. Estudio de ROS y navegación autónoma

## 7.1 Introducción a las simulaciones ROS

Como ya se comentó en los objetivos de este proyecto, primero de todo, se va a trabajar con simulaciones debido a que el robot real puede proporcionar problemas inoportunos, para evitar esto, se realizan diferentes pruebas simuladas con TurtleBot3 Waffle Pi.

Para tener un contexto de como se trabaja con este tipo de simulaciones, y conocer el entorno y las posibilidades que nos ofrece ROS y TurtleBot3, se explican diferentes ejemplos de simulaciones para así tener una base, y a partir de ahí, saber y tener una idea de como se va a trabajar más adelante.

Un dato importante que saber es que en todo momento se va a trabajar con la herramienta denominada Gazebo y RVIZ. Para empezar a trabajar con dichas simulaciones se hace la siguiente instalación de simulaciones proporcionada por Robotis.

Antes de realizar las siguientes instrucciones, primero se deben de instalar todos los paquetes necesarios y el tipo de ROS con el que se va a trabajar. Para ello, se deberán de seguir los pasos detallados que están proporcionados en el anexo de este proyecto.

```
1. $ cd ~/catkin_ws/src/

2. $ git clone -b melodic-devel https://github.com/ROBOTIS-
   GIT/turtlebot3_simulations.git

3. $ cd ~/catkin_ws && catkin_make
```

A continuación, se detalla con que modelo de TurtleBot3 se quiere trabajar, y posteriormente cargamos el mapa y el entorno. Este mapa está ya creado, únicamente lo que se hace es cargarlo para que se abra en Gazebo.

```
1. $ export TURTLEBOT3_MODEL=waffle_pi
2. $ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

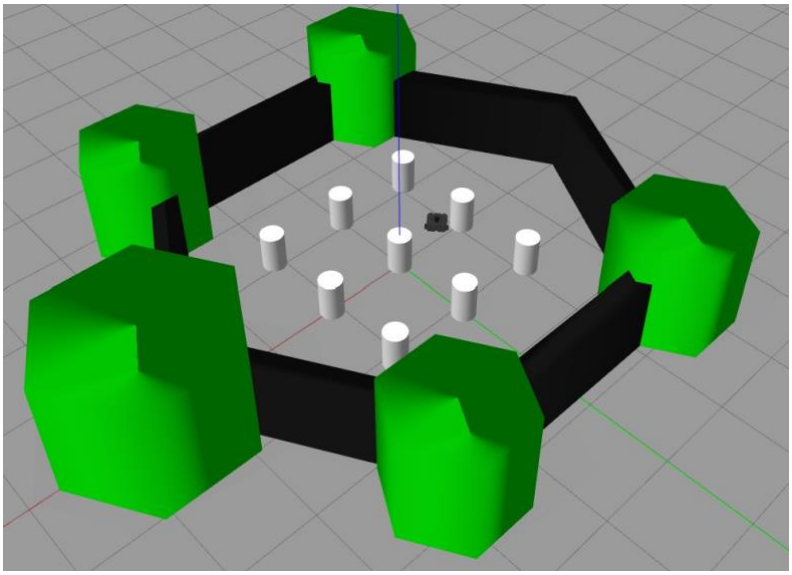

Figura 7. 1 Entorno mapa Gazebo

Fuente: Elaboración propia

Se puede trabajar tele operando con diferentes teclas del teclado del portátil, en este caso, se lanza el siguiente comando proporcionado por Robotis.

```
1. $ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Con el programa Rviz ya comentado anteriormente, se puede visualizar los datos recogidos por el TurtleBot3.

```
1. $ roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
```

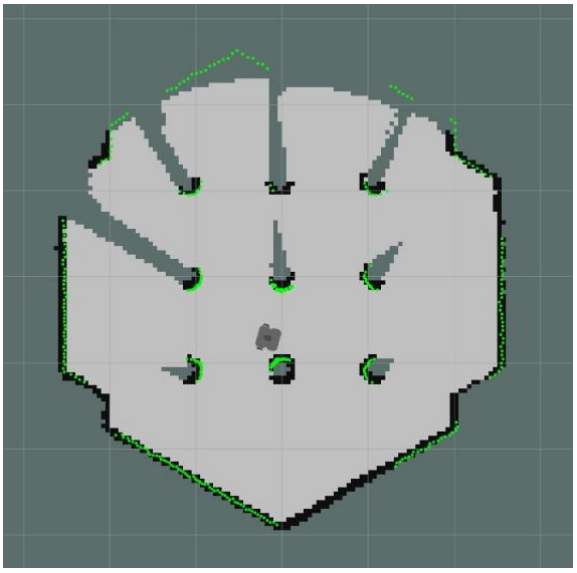

Figura 7. 2 Investigando entorno mapa RViz

Fuente: Elaboración propia

Como se observa en la figura 7.2, a medida que el robot va avanzando respecto la posición inicial, éste va reconociendo el entorno por el cual se va moviendo, hasta que finalmente lo tiene todo identificado.

### 7.1.1 Simulación SLAM

Con la simulación SLAM se pueden seleccionar y crear diferentes puntos y modelos de robot en un mundo virtual que hayamos creado.

A continuación, se observará un pequeño resumen de cómo se crea y que script hace falta para guardar el mapeado por donde el robot ha paseado.

Primero de todo, se inserta el modelo con el que se trabaja y se carga el entorno virtual por donde se quiere que el robot posteriormente grabe el mapeado.

1. $ export TURTLEBOT3_MODEL=waffle_pi
2. $ roslaunch turtlebot3_gazebo turtlebot3_world.launch

Se abre una nueva terminal y se ejecuta el nodo SLAM como se indica a continuación:

1. $ export TURTLEBOT3_MODEL=waffle_pi
2. $ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping

En este caso, se abre otra terminal y en esta fase únicamente se explica el funcionamiento de lo que se puede llegar a hacer con ROS. El siguiente paso se ejecuta un nodo de tele operación para controlar el robot para finalmente grabar el mapeado.

1. $ export TURTLEBOT3_MODEL=waffle_pi
2. $ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

Una vez, se ha recorrido e identificado todo el mapeado, se procede a grabar el mapa.

```
3. $ rosrun map_server map_saver -f ~/map
```

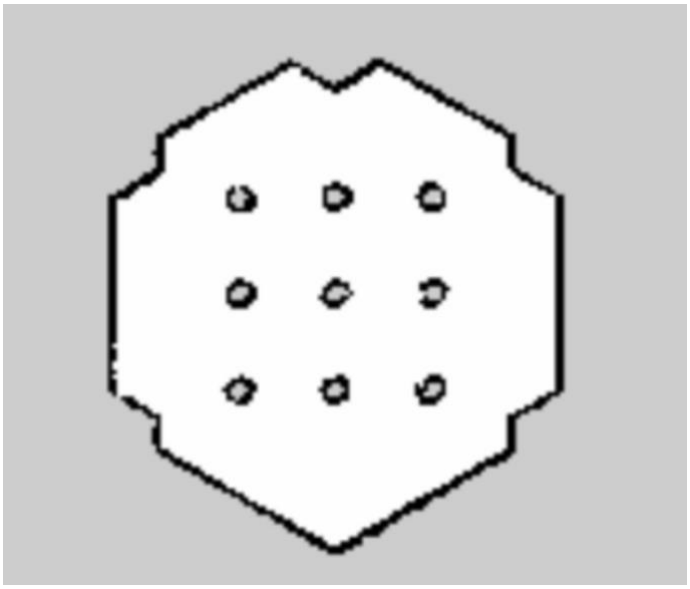

Figura 7. 3 Entorno mapa RViz

Fuente: [29]

### 7.1.2 Simulación de navegación

Y como último apartado sobre la introducción al mundo de la simulación a través de la herramienta Gazebo, se verá el nodo de navegación.

Primero de todo, se inserta el modelo con el que se trabaja y se carga el entorno virtual por donde queremos que el robot posteriormente grabe el mapeado.

1. $ export TURTLEBOT3_MODEL=waffle_pi
2. $ roslaunch turtlebot3_gazebo turtlebot3_world.launch

Se abre un nuevo terminal donde se ejecuta el nodo de navegación.

1. $ export TURTLEBOT3_MODEL=
2. $ roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml

Se indica la posición inicial donde se quiera que el robot comience su recorrido por el mapa.


Figura 7. 4 Entorno RViz posición inicial

Fuente: [29]

Una vez, se tiene el robot colocado en la posición deseada establecemos el objetivo de navegación. Para ello, en la barra de herramientas de la parte de arriba de la herramienta RVIZ se observa una flecha de color rojo donde pone "2D Nav Goal". Esta flecha se coloca en aquella posición donde quieras que el robot vaya, y una vez se coloca en el mapa, el robot irá hacia la flecha de forma autónoma.

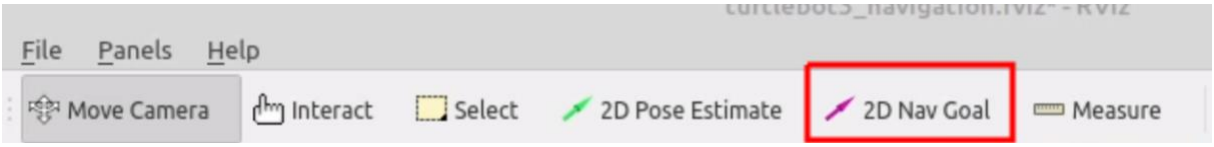


Figura 7. 5 Entorno RViz posición final

Fuente: [29]

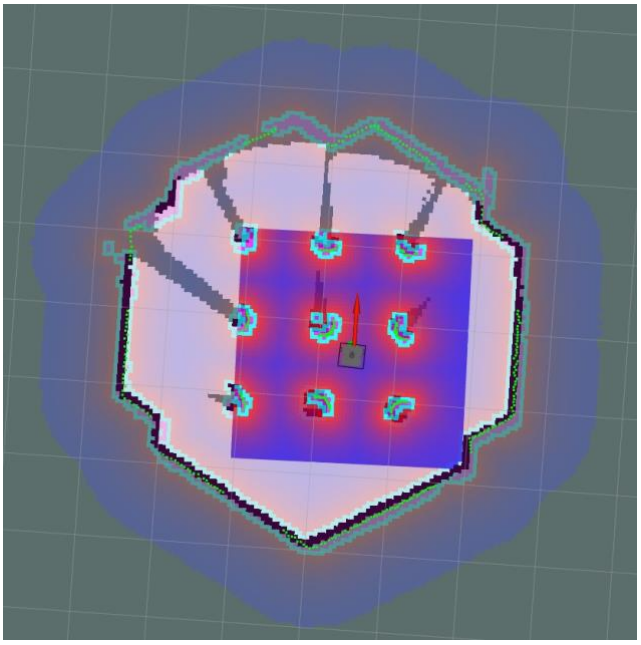

Figura 7. 6 Entorno RViz navegando hasta posición final

Fuente: Elaboración propia

## 7.2 Navegación autónoma del TurtleBot3

Cuando ya se tiene un conocimiento básico sobre como manejar ROS y las diferentes herramientas que nos puede llegar a proporcionar, ya se puede indagar en el tema principal de este proyecto. Para ello, para ese aprendizaje autónomo a partir del reconocimiento de patrones y redes neuronales se ha tenido que descargar Anaconda 5.2.0 para el uso del Python de la versión 2.7.

Para comprobar que se trabaja con lo mencionado en el párrafo anterior, se hace lo siguiente:

1. $ bash Anaconda2-5.2.0-Linux-x86_64.sh
2. $ source ~/.bashrc
3. $ python -V

Si todo se ha instalado correctamente, la terminal devolverá: Python 2.7.15 :: Anaconda, Inc.

Se instalan los paquetes de dependencia ROS que proporciona Robotis.

1. $ pip install msgpack argparse
2. $ pip install -U rosinstall empy defusedxml netifaces

Antes de realizar las dos instalaciones anteriores se debe de hacer una actualización del *pip*. Para ello, se habrá de hacer lo siguiente:

1. $ pip install –upgrade pip

Se instala TensorFlow:

1. $ pip install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-1.8.0-cp27-none-linux_x86_64.whl

Se instala el código abierto keras:

1. $ pip install keras==2.1.5

Se instalan los paquetes de machine learning proporcionados por Robotis, para así poder trabajar con ello, y tener los diferentes mapeados y códigos para practicar, estudiar y poner en práctica el aprendizaje autónomo

1. $ cd ~/catkin_ws/src/
2. $ git clone https://github.com/ROBOTIS-GIT/turtlebot3_machine_learning.git
3. $ cd ~/catkin_ws && catkin_make

Ya que el paquete numpy tiene una gran relevancia y para comprobar que no existe ningún problema en su instalación, se va a desinstalar del todo, para posteriormente volver a instalarlo y asegurarnos de que está completo y listo para su ejecución.

1. $ pip uninstall numpy
2. $ pip show numpy
3. $ pip install numpy pyqtgraph

Una vez realizados estos pasos anteriores, y como se comentó en el apartado de los objetivos de este proyecto, primero de todo, se trabaja de forma virtual. Una vez, se tenga el objetivo cumplido de forma virtual se avanza al siguiente paso, este paso es manejar el robot de forma real con sus respectivas configuraciones y requisitos que se necesite. A continuación, se muestra los pasos que se han seguido para lograr el objetivo de forma virtual.

## 7.2.1 Estudio del aprendizaje autónomo utilizado

### 7.2.1.1 Método de entrenamiento

Para trabajar con redes neuronales se debe de aplicar un modelo de entrenamiento. Se puede entrenar el robot de muchas maneras, en este caso, se ha decidido entrenar al TurtleBot3 con el algoritmo DQN.

El algoritmo DQN es una inteligencia artificial diseñada para realizar imitaciones del comportamiento humano.

Este tipo de arquitectura, en este caso, utiliza el aprendizaje por refuerzo y diferentes redes neuronales profundas para hacer predicciones y actualizaciones de los cálculos de Q futuros.

A continuación, en los próximos apartados se explica detenidamente todo aquello que necesita el algoritmo DQN para trabajar como es debido.

#### 7.2.1.1.1 Redes neuronales y reconocimiento de patrones

Como ya se sabe desde un principio, se pretende manejar el robot de una forma autónoma, pero haciendo uso de redes neuronales. Para trabajar con las redes neuronales se utilizan dos bibliotecas de código abierto denominadas TensorFlow y Keras, explicadas detenidamente en el apartado de marco conceptual de este proyecto.

En este caso, el reconocimiento de patrones se realiza a través del modelo neuronal, el cual está construido con Keras. Como ya se ha comentado en el apartado de cómo se han utilizado las redes neuronales en el código diferenciadas por capas. Estas capas están diseñadas para reconocer patrones en los datos de entrada y aprender a realizar a tomar decisiones y predicciones basados en los patrones.

Para el programa utilizado para el aprendizaje de propagación posterior para la navegación autónoma, y haciendo uso de redes neuronales para lograr el objetivo, se va a explicar a continuación como se ha desarrollado el proceso de redes neuronales y cómo se han establecido.

Primero de todo, las redes neuronales utilizadas se dispersan en diferentes capas, se diferencian de la siguiente manera:

1. La primera capa es la de entrada, la cual recibe el estado como entrada.

Capas intermedias:

2. Hay una primera capa oculta que contiene un número de redes neuronales, y utiliza la función de activación *leaky relu* Esta función de activación introduce la no linealidad en la red neuronal. Existen varios tipos de funciones de activación para las redes neuronales, algunas de ellas son *relu*, *tanh*, *leaky relu*, y muchas otras, pero con las que no se ha experimentado en este proyecto. En este caso, se ha decidido trabajar con la función de activación denominada leaky relu. Leaky relu trabaja de una manera que soluciona el problema de las neuronas muertas que pueden surgir con otras funciones de activación. El hecho de aplicar una pendiente en valores negativos provoca que la información fluya incluso cuando hay valores negativos de entrada, se aprovecha esa labor para mejorar el entrenamiento y el rendimiento de la red neuronal utilizada.
3. Una segunda capa oculta que contiene otro número de redes neuronales, y utiliza la función de activación *leaky relu*. En esta segunda etapa, se agrega una capa dropout, esta regulariza el modelo apagando aleatoriamente neuronas durante el entrenamiento para que no se produzca un sobreajuste, provocando una mejoría de la red neuronal.
4. Una tercera capa oculta que contiene otro numero de redes neuronales, y utiliza la función de activación *leaky relu*. En este caso, también se hace uso del dropout para regularizar el entrenamiento y evitar el sobreajuste.

Capa de salida:

5. Y una capa de salida, que representa el tamaño de salida del modelo, y el número de neuronas está determinado por el tamaño de la acción.

A continuación, en la siguiente página se muestra una representación gráfica de la estructura de las redes neuronales utilizadas:

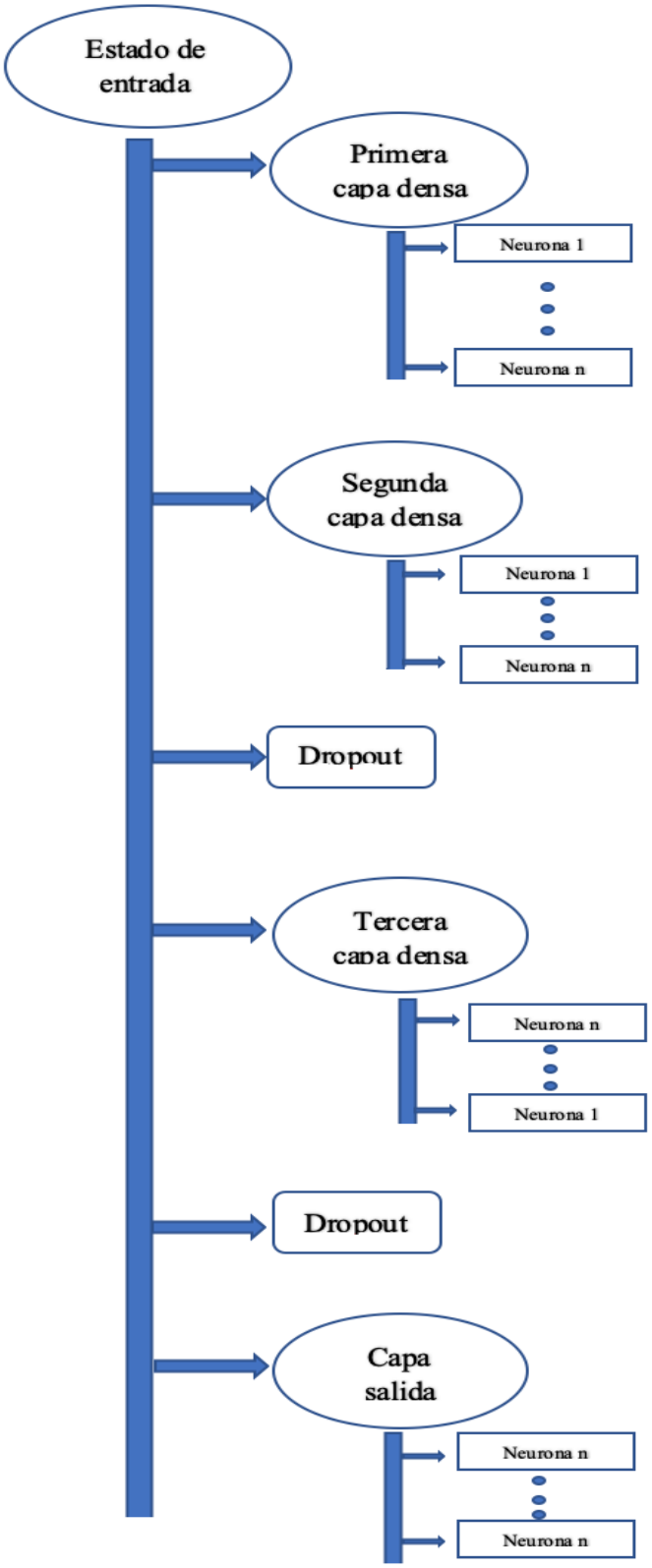


Figura 7.7 Esquema redes neuronales

Fuente: Elaboración propia

#### 7.2.1.1.2 Método para realizar la acción

En este apartado se ha hecho un estudio de cual puede ser el mejor método para utilizar para el objetivo de este proyecto. Se han tenido en cuenta 2 métodos, los llamados Epsilon-greedy y Softmax.

**Epsilon-greedy**

En este método se utiliza un valor de épsilon, este valor se ajusta al entorno y al modelo con el cual se quiere trabajar.

La fórmula utilizada para cada acción Z, para este método se puede expresar de la siguiente manera:

- Si hay una selección aleatoria llamada exploración, la probabilidad de elegir Z es con la siguiente fórmula:
  - (épsilon / número de acciones totales)
- En cambio, si se elige la mejor acción conocida hasta ese momento, este proceso se denomina explotación, la probabilidad de elegir la acción Z, si Z fuese la mejor opción es la siguiente:
  - (1 – epsilon), se elige la acción con el valor Q máximo para explotar el conocimiento.

Y si Z no es la mejor acción:

- (épsilon / número de acciones totales)

Es decir, épsilon es un valor que se proporciona entre 0 y 1, y que representa una posibilidad de elección de una acción aleatoria en lugar de aquella acción conocida que pueda ser mejor. Si se proporciona un valor bajo de épsilon, se indica una mayor tendencia hacia la explotación, por otro lado, si se proporciona un valor de épsilon elevado indica una mayor tendencia a la exploración

**Softmax**

Por otro lado, en este método, todas las acciones tienen cierta probabilidad de ser seleccionadas a través de los valores Q. Todas aquellas acciones que tengan los valores Q más elevados tienen una mayor probabilidad de ser elegidas, pero las acciones que tienen un valor Q más bajo también tienen una pequeña posibilidad de ser elegidas.

Además, en el método Softmax se utiliza el parámetro de temperatura que se utiliza para controlar el nivel de exploración. Un valor de temperatura que sea más alto provoca una distribución de oportunidades uniforme, hecho que genera una exploración más equilibrada. A lo largo del entrenamiento, cuando decae el valor de la temperatura disminuye la exploración e incrementa la explotación.

Cuando ya se tiene el conocimiento suficiente, y se han realizado diferentes pruebas para averiguar cual es el mejor método para utilizar para este proyecto, finalmente, el método a utilizar para la decisión de acciones es el método de Epsilon-greedy.

Este método da una mayor fiabilidad, ya que es más determinista a la hora de seleccionar las acciones, debido a que escoge aquella acción con el valor máximo de Q, y para este proyecto es la óptima.

#### *7.2.1.1.3* Hiperparámetros

Como ya se ha comentado en el apartado anterior, se utiliza el método Epsilon-greedy para la realización de este proyecto. Al utilizar este método, se hace uso de un seguido de parámetros, en este caso denominados hiperparámetros.

Ya que después de realizarse un estudio, y elegir este método para trabajar, es necesario implementar los hiperparámetros que se pueden observar en la tabla siguiente, con sus respectivas definiciones:

| Hiperparámetros | Definición de los hiperparámetros |
|---|---|
| Paso del episodio | Número máximo de pasos que se permiten en un episodio. |
| Objetivo de actualización | Determina la frecuencia con la que se actualiza la red neuronal objetivo. La red neuronal objetivo es una copia de la red principal que se utiliza para calcular los valores de Q durante el entrenamiento. |
| Factor de descuento | Factor de descuento utilizado para calcular el retorno descontado en el aprendizaje. Determina la importancia de las recompensas futuras en comparación con las recompensas inmediatas. |
| Tasa de aprendizaje | Establece la tasa de aprendizaje utilizada en la optimización para actualizar los pesos de la red neuronal. |
| Épsilon | Valor inicial de la exploración (epsilon greedy). La exploración es la estrategia de tomar acciones aleatorias con el fin de descubrir nuevos conocimientos. |
| Caída de épsilon | Determina la tasa a la que se reduce el valor de exploración a lo largo del tiempo. La reducción que se produce gradualmente permite que avance más en el entrenamiento. |
| Épsilon mínima | Establece un valor mínimo que pueda alcanzar épsilon. Cuando épsilon llega a este valor el modelo se centra más en la explotación. |
| Tamaño del lote | Indica el tamaño de los lotes de muestras utilizadas para el entrenamiento de la red neuronal. Aprendizaje en lotes. |
| Inicio tren | Representa el número mínimo de muestras en la memoria de reproducción antes de comenzar el entrenamiento. La memoria de reproducción se utiliza para almacenar y recuperar muestras de experiencia pasada. |

Tabla 7.1 Tabla hiperparámetros

Es muy importante la implementación de estos hiperparámetros para que el modelo funcione correctamente. Además, se les ha de aplicar aquellos valores que se crean más optimas para la hora de realizar el estudio. Cabe recalcar que para cada entorno se van a necesitar unos valores u otros, esto depende de los obstáculos y de hacer un estudio de pruebas para ver que valores pueden ser los óptimos.

#### 7.2.1.1.4 Recompensas

Lo que se pretende es que waffle pi sea capaz de aprender autónomamente el entorno que le rodea, y los posibles obstáculos que le pueden hacer frente. ¿Cómo el robot es capaz de saber si va por buen camino o no? Para decirle de alguna manera al robot los puntos a evitar y los que no, se le otorga un seguido de recompensas, en base estas recompensas el robot obtiene dicho reconocimiento del terreno, dando así lugar a que finalmente el robot sea capaz de desplazarse de forma autónoma evitando obstáculos por todo el mapeado.

Visto lo comentado en el párrafo anterior, no es un proceso de aprendizaje instantáneo, sino que requiere de un tiempo determinado y de una gran cantidad de episodios hasta que el robot sea capaz de desplazarse sin chocarse.

Es decir, cuando el robot colisiona en el código programado se le debe de atribuir una recompensa negativa, para que éste sea consciente de que por ahí no puede pasar o acceder. Por otro lado, en el entorno 3D y una vez lanzado el código, aparece un objetivo en pantalla, este objetivo debe de ser alcanzado por el robot, cuando el Waffle Pi llega al punto deseado se le otorga una recompensa positiva, para que éste sepa que va por buen camino. A continuación, se observa que cantidad de puntos se le proporciona a cada uno, y el mensaje que sale cuando el robot hace la función correspondiente.

```
rospy.loginfo("Colision!!! Por favor, empeice de nuevo!")
reward = -200
```

Figura 7.8 Puntuación colisión

Fuente: Elaboración propia

```
rospy.loginfo("Enhorabuena!!! Puede continuar navegando")
reward = 200
```

Figura 7.9 Puntuación objetivo cumplido

Fuente: Elaboración propia

Además de recibir un valor X de recompensa por alcanzar el objetivo o colisionar, el robot recibe recompensas por cada acción que realiza. Es decir, si durante la navegación el robot se desplaza por un entorno, el cual anteriormente se le han atribuido recompensas negativas debido a que se ha colisionado cerca, el Waffle Pi va recibiendo y acumulando todas estas recompensas, quiere decir, las recompensas son acumulativas. Esto significa, que desde que el robot inicia su trayecto desde el eje 0,0 hasta que colisiona o alcanza el objetivo, por cada acción que se realiza, se van almacenando todas las recompensas.

#### 7.2.1.1.5 Cálculo del Q-value

El Q-value es una medida para expresar cuanto vale realmente la pena tomar una determinada decisión, en este caso una acción en un estado sirve de guía para la toma de decisiones.

El valor de Q en este proyecto se utiliza durante la fase de entrenamiento para actualizar al robot con el que se está trabando y el objetivo que se tiene que alcanzar. Mientras que el Waffle Pi está entrenando, se recoge una muestra de la memoria de la experiencia que está obteniendo el robot, a partir de esto, se calcula el valor de Q para cada estado y acción en la muestra que se acaba de comentar.

Finalmente, este valor de Q calculado anteriormente se utiliza para volver actualizar los valores, pero en este caso los valores de los diferentes estados y acciones respecto al robot que está entrenando.

En este caso, este valor Q se calcula como el máximo valor que puede llegar a tener Q para el próximo estado y la suma de la recompensa que se tiene en ese momento, todo esto ponderado por el factor de descuento, este factor de descuento se explica detenidamente en la tabla 7.1.

### 7.2.2 Navegación autónoma en entorno plaza

Una vez se ha realizado el estudio pertinente de como realizar el código y la elección de todos sus parámetros, se da comienzo al estudio de las simulaciones.

Inicialmente, se trabaja en un entorno inicialmente proporcionado por Robotis, el cual se puede observar en la siguiente figura.

1. $ export TURTLEBOT3_MODEL=waffle_pi
2. $ roslaunch turtlebot3_gazebo turtlebot3_stage_2.launch

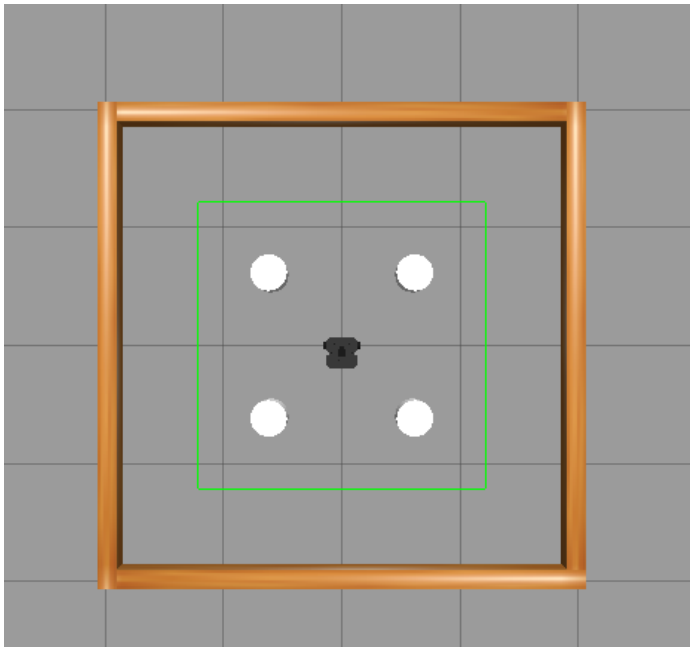

Figura 7.10 Entorno plaza en Gazebo para practicar

Fuente: Elaboración propia

Además de visualizar el mapa con el entorno Gazebo, se proporcionan unas gráficas proporcionadas por Robotis donde se ve representado el valor total de las recompensas y el Q-value, para así tener una idea más visual de como el robot está trabajando. Para ello, se abre una nueva terminal y se escribe el siguiente comando:

1. $ roslaunch turtlebot3_redes_neuronales gráfica.launch

Con la acción anterior, además de ver los resultados en dos gráficas posibles, se incorpora las acciones que puede hacer el robot como se muestra en la siguiente figura.

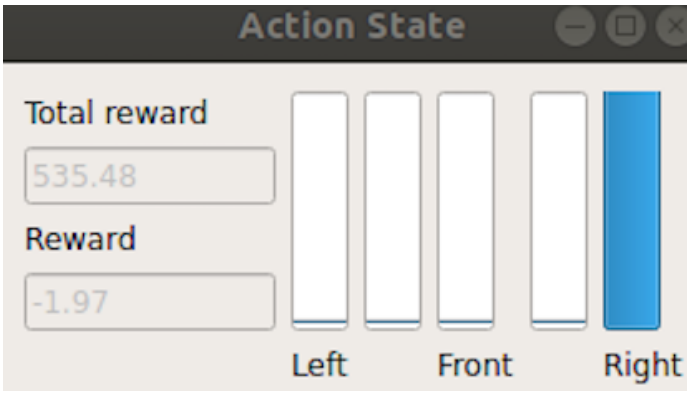


Figura 7.11 Acción del Waffle Pi

Fuente: Elaboración propia

Con esta imagen de las acciones que está realizando el robot, se puede tener el conocimiento de que acción está realizando y que recompensa está recibiendo en cada momento de su interacción con el entorno y así tenerlo controlado.

Para el aprendizaje automático es importante establecer un seguido de hiperparámetros. También conocidos como hiperparámetros de las redes neuronales, estos parámetros se establecen para tener la posibilidad de controlar el entrenamiento que está experimentando el robot. Sin esta clase de parámetros, el TurtleBot3 no seria capaz de aprender el entorno que le envuelve, dando lugar a que seria imposible lograr el objetivo deseado.

Para que el robot comience el proceso de aprendizaje, se lanza en una nueva terminal el siguiente código:

1. $ roslaunch turtlebot3_redes_neuronales  turtlebot3_redes_neuronales_1.launch

Una vez introducidos estos comandos en los diferentes terminales, el robot se va a poner en marcha y se va a comenzar a mover de manera autónoma. Además, se van a observar un seguido de gráficas que se van a comentar y se van a separar en diferentes etapas. De esta manera, se puede entender el comportamiento y el aprendizaje que está obteniendo el robot en la simulación, y así poder ver su evolución de una forma más visual.

Primera etapa

Tras varios episodios e intentos del robot y con poco terreno reconocido, es poco probable que el Waffle Pi llegue al objetivo deseado. Esto quiere decir, que el robot está colisionando de forma repetitiva con las paredes o los cilindros que contiene el mapa. Este comportamiento al inicio es normal, ya que el robot se encuentra en un entorno completamente desconocido, y tiene que ir aprendiendo lo que es correcto y lo que no, para ello, se establecen las recompensas ya comentadas anteriormente para distinguir estas dos opciones

Antes de seguir avanzando con la explicación, se realiza un pequeño inciso para hacer una breve explicación de lo que son los episodios. Un episodio, en este caso, es el período de tiempo que el robot está circulando por el mapa, dicho episodio finaliza cuando el robot colisiona, como se va a poder ver en esta primera etapa. Sin embargo, si el robot no colisiona en ningún momento y durante este período de tiempo solo consigue los objetivos, no se cambia de episodio, esto quiere decir que el robot ya no está colisionando y el aprendizaje está funcionando, ya que el robot ha conseguido desplazarse de forma continua sin chocar con ningún obstáculo.

En la figura 7.11 se observa como el robot no es capaz de alcanzar una recompensa grande, ya que cada vez que colisiona se le quitan una cantidad de puntos.


Figura 7.12 Recompensa total después de 50 episodios

Fuente: Elaboración propia

En la figura 7.12 se observa como prácticamente el Q-value inicial es nulo o tiene números claramente negativos. A lo largo de estos episodios, se podrá observar una clara evolución de aprendizaje.

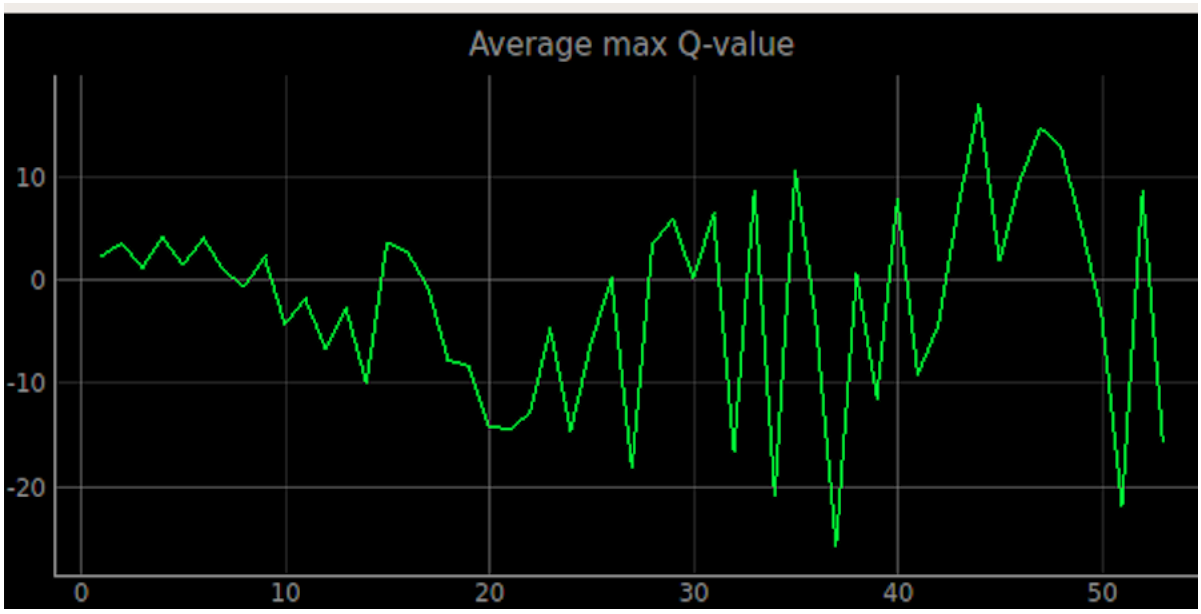


Figura 7.13 Q-value después de 50 episodios

Fuente: Elaboración propia

Si se mira el transcurso de tiempo, casi al inicio de la ejecución del programa se observa que hay un episodio nuevo cada un minuto o inclusos segundos. Esto se puede apreciar en las figuras 7.13 y 7.14, viendo así que del episodio 47 al 49 solo hay una diferencia de 28 segundos.

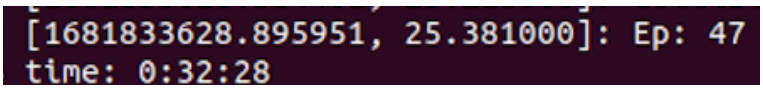


Figura 7.14 Tiempo episodio 47

Fuente: Elaboración propia

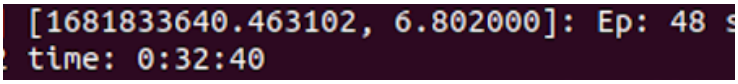


Figura 7.15 Tiempo episodio 48

Fuente: Elaboración propia

Segunda etapa

En esta segunda etapa se empieza a observar un cambio notable respecto a la primera etapa, se aprecia en la figura 7.15 como a partir del episodio 80 aproximadamente el total de las recompensas comienza a ser ya positivo de una forma constante. Esto significa, que el robot ya comienza a alcanzar más objetivos deseados y comienza a no colisionar tanto con el entorno.

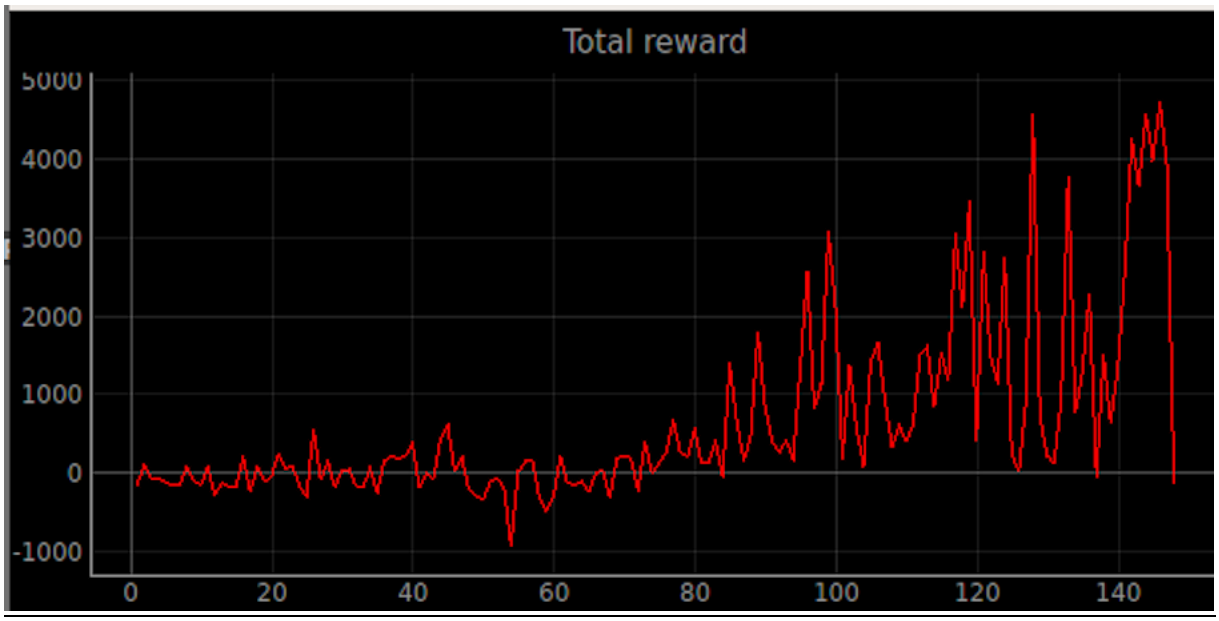


Figura 7.16 Recompensa total después de 140 episodios

Fuente: Elaboración propia

En la figura 7.16 se puede observar que el Q-value empieza a ser positivo de una forma permanente. Este cambio, se puede empezar a apreciar a partir del episodio número 100, donde de una manera significativa el valor comienza a aumentar, así, dando ya indicios de que el robot empieza a no colisionar.

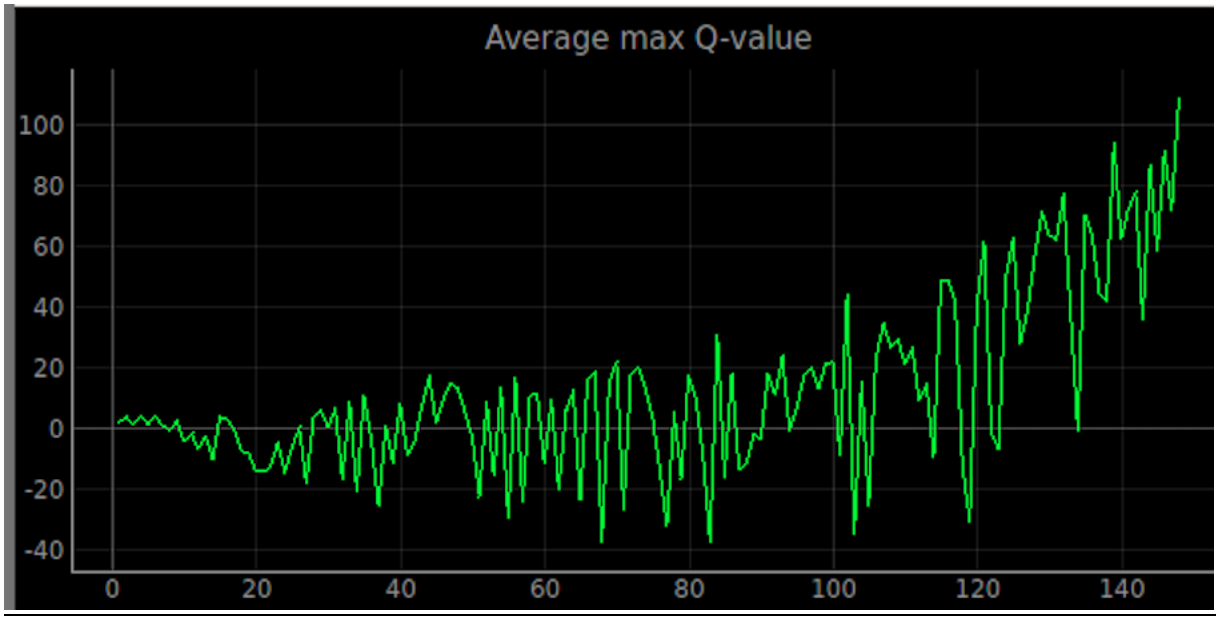


Figura 7.17 Q-value después de 140 episodios

Fuente: Elaboración propia

En este momento, ya se comienza a notar la diferencia de tiempo entre episodio y episodio, habiendo un margen de varios minutos.

```
[1681843286.537079, 146.174000]: Ep: 146
0.23 time: 3:13:26
```

Figura 7.18 Tiempo episodio 146

Fuente: Elaboración propia

```
[1681843547.775735, 147.212000]: Ep: 147
0.23 time: 3:17:47
```

Figura 7.19 Tiempo episodio 147

Fuente: Elaboración propia

Tercera etapa

En esta última etapa ya se puede observar un gran progreso respecto el inicio, se observa claramente como la puntuación recogida es completamente positiva llegando a alcanzar valores de hasta 5000 puntos, y un Q-value superior a 200. Esto quiere decir, que el robot ya prácticamente no colisiona y únicamente consigue los objetivos. En la figura 7.19 se aprecia claramente el recorrido que ha ido teniendo el robot, durante aproximadamente 12 horas. Al

inicio el valor de la recompensa no es muy elevado debido a lo que se ha comentado anteriormente, que el robot colisiona a menudo, pero a partir del episodio 85 se puede observar como comienza a conseguir puntuaciones más altas gracias al alcance de los objetivos obtenidos.

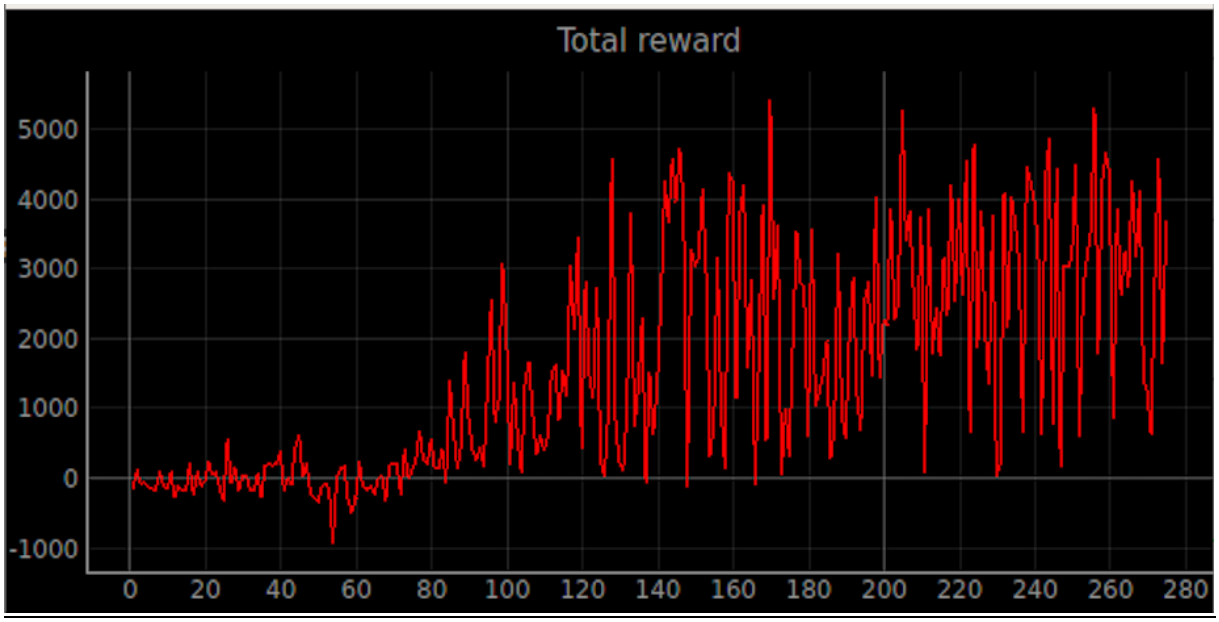


Figura 7.20 Recompensa total después de 280 episodios

Fuente: Elaboración propia

Lo mismo pasa en la figura 7.20, donde se observa que el Q-value en un inicio es prácticamente nulo o incluso negativo, hasta que en la etapa 90 aproximadamente este valor comienza a aumentar, y a partir de aquí, el valor ya es siempre positivo llegando a alcanzar y superar una media bastante elevado respecto a la del inicio.

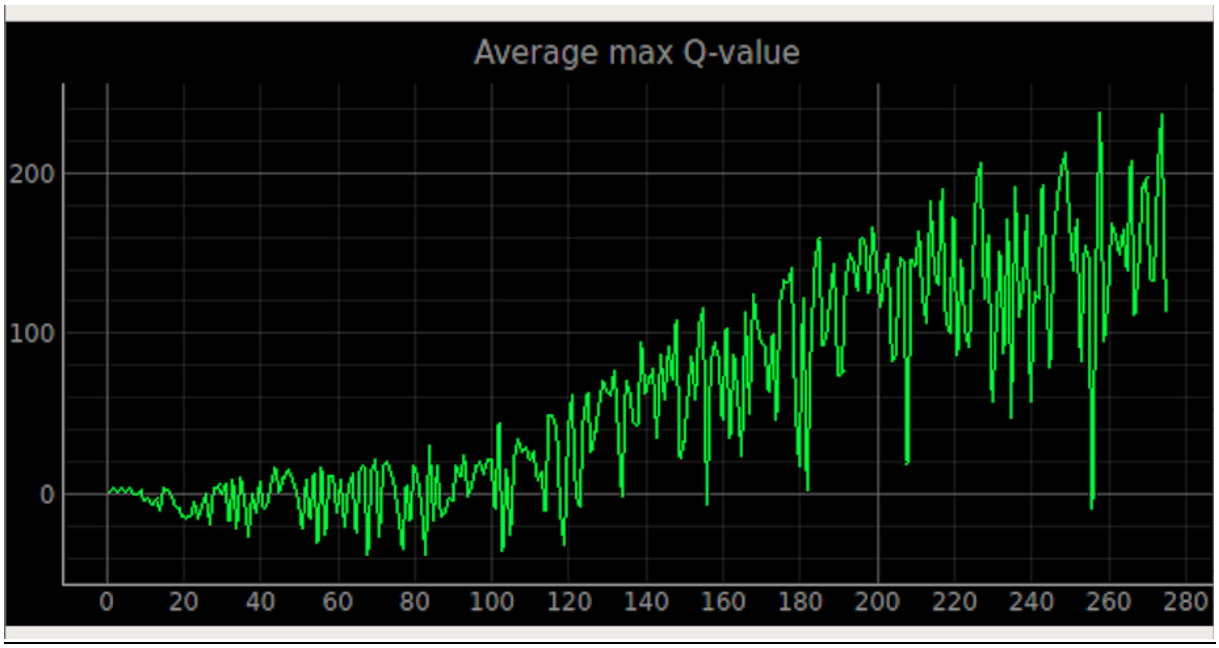


Figura 7. 21 Q-value después de 280 episodios

Fuente: Elaboración propia

En esta última muestra de diferencia de tiempo entre episodios, se observa que el transcurso de tiempo entre los episodios es bastante mayor respecto al de la primera etapa, habiendo una diferencia de 12 minutos entre el episodio 275 y 276.

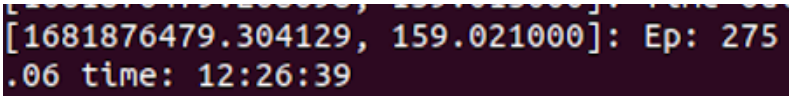


Figura 7.22 Tiempo episodio 275

Fuente: Elaboración propia

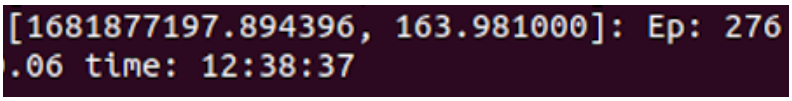


Figura 7.23 Tiempo episodio 276

Fuente: Elaboración propia

Si se observa el progreso que tiene el robot detenidamente, desde la gráfica inicial hasta la última gráfica de este documento, se observa claramente una mejoría. Como el Turtlebot3 al inicio del proceso se desplaza confundido, sin tener muchos conocimientos de su entorno, así colisionando repetitivamente con los mismos obstáculos, hasta que finalmente por experiencia

propia el robot ya tiene identificado y guardado los puntos por los cuáles no debe pasar, evitándolos así y cumpliendo los objetivos.

### 7.2.3 Navegación autónoma en entorno laberinto

En el apartado anterior, se ha visto trabajar al Waffle Pi de forma autónoma y aprendiendo en base a errores y a una cantidad de tiempo considerable el entorno que le rodea. Este entorno visto en la figura 7.10, como ya se comentó anteriormente, estaba proporcionado por Robotis.

En este apartado, se va a trabajar con un nuevo entorno creado para que el TurtleBot3 tenga una mayor dificultad en conseguir y alcanzar todos los objetivos propuestos, y de esta manera, poder ver como trabaja y como se desenvuelve en un entorno diferente y poder hacer una breve comparativa con el anterior. Dada la diferencia de obstáculos entre un mapa y otro, se puede anticipar que el tiempo de aprendizaje en este caso será bastante mayor respecto el anterior mapa.

Primero de todo, se ha elaborado un mapa en el entorno Gazebo, dicho mapa contiene diferentes adversidades, así proporcionando un poco más de complejidad al objetivo final que es lograr los diferentes objetivos. A continuación, se proporciona un plano en el cuál se puede observar las diferentes habitaciones, con sus respectivas puertas para entrar y salir, y diferentes obstáculos entremedios para aportar una mayor dificultad.

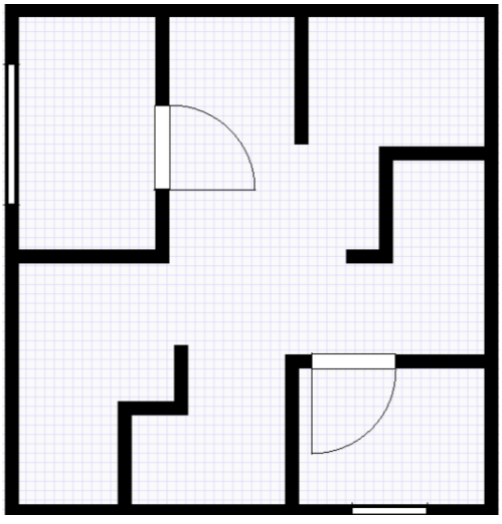

Figura 7.24 Plano entorno laberinto en Gazebo

Fuente: Elaboración propia

Se ha elaborado un código para la elaboración del plano de la figura 7.24. Pero, para entenderlo mejor y tener una referencia más visual se ha insertado el plano que se observa en la figura anterior.

Una vez, se ha elaborado el mapa se procede a entrar en el entorno de Gaebo para visualizar el plano en un formato 3D. Para entrar en el mapa de Gazebo, se han de realizar los siguientes pasos:

- Se abre una terminal y se escribe lo siguiente:
  1. $ roscore
- Se abre una nueva terminal y se lanzan los siguientes comandos:
  1. $ export TURTLEBOT3_MODEL=waffle_pi
  2. $ roslaunch turtlebot3_redes_neuronales  turtlebot3_redes_neuronales_1.launch

Cuando se hayan lanzado los comandos anteriores, en el orden como se ha comentado, se observa el contenido que se observa en la siguiente figura.

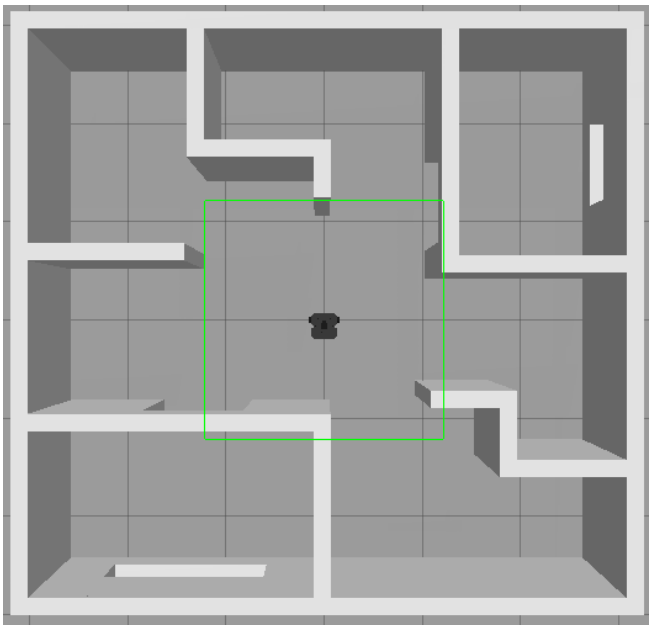

Figura 7.25 Entorno 3D laberinto

Fuente: Elaboración propia

Respecto a la figura 7.9, se aprecia que el mapa ha cambiado considerablemente, así viendo que hay más dificultad en este nuevo. A continuación, se proporciona otra imagen del entorno desde otra perspectiva para visualizar mejor las puertas y las ventanas que contiene el nuevo mapa.

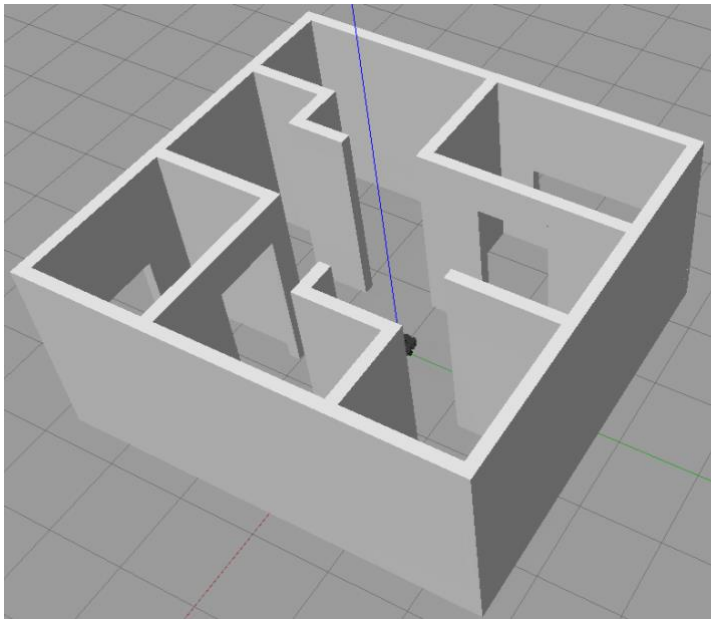

Figura 7.26 Entorno 3D laberinto desde otra perspectiva

Fuente: Elaboración propia

Una vez creado el mapa deseado para esta ocasión, se puede lanzar el código correspondiente para que el Waffle Pi comience su aprendizaje autónomo en un nuevo entorno desconocido. Pero, antes de lanzarlo, se van a comentar varios aspectos diferentes respecto al entorno de la plaza.

El proceso y el funcionamiento para este entorno es el mismo que para el entorno de la plaza, es decir, trabaja con episodios, estableciendo objetivos y otorgándole una recompensa ya sea negativa o positiva dependiendo de lo que el Waffle Pi haya logrado

A continuación, se observa el valor de las recompensas que se la han otorgado, en esta ocasión coinciden con el caso anterior.

Una vez introducido el mapa y el valor de las recompensas, se lanzan los comandos necesarios para que el TurtleBot3 comience su aprendizaje en este nuevo entorno creado denominado laberinto

- Primero de todo, se abre una terminal donde se escribe lo siguiente:

  1.$ roscore
- Segundo, se abre una nueva terminal donde se abre el mapa creado en el entorno 3D Gazebo:

2. $ roslaunch turtlebot3_gazebo turtlebot3_stage_1.launch

- Tercero, se abre otra nueva terminal donde se lanza el comando para ver las gráficas de las recompensas y del Q-value

3. $ roslaunch turtlebot3_dqn result_graph.launch

- Por último, se abre una nueva terminal, en la cual se va a lanzar el código para que el Waffle Pi comience su navegación por el entorno desconocido.

4. $ roslaunch turtlebot3_redes_neuronales  turtlebot3_redes_neuronales_1.launch

Una vez realizados los pasos anteriores, el robot ya se estará desplazando de forma autónoma por el laberinto. Además, en este caso, se va a lanzar un comando donde se puede observar lo que el robot tiene delante, es decir, vamos a activar la cámara que contiene el robot en la simulación para tener una visualización, y poder ponernos en la situación del robot en todo momento. Para ello, se lanza el comando siguiente:

5. $ rqt_image_view

Cuando ya se ha realizado el proceso anterior, aparece la siguiente ventana:

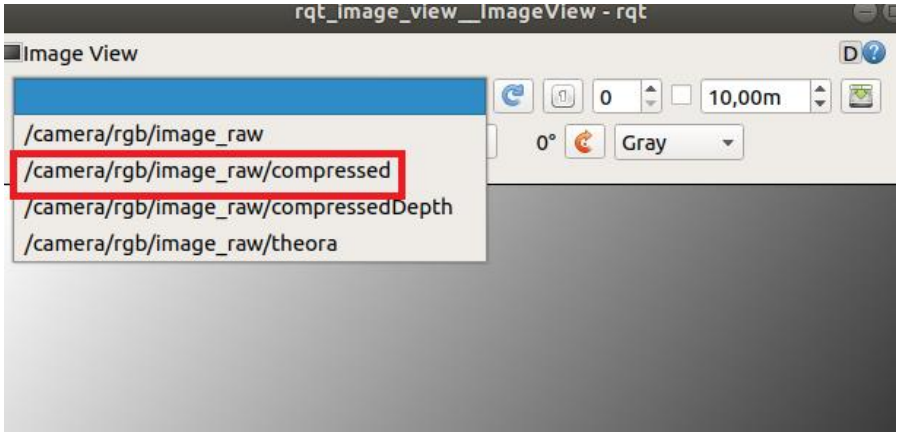


Figura 7.27 Selección del tipo de cámara

Fuente: Elaboración propia

Como se observa en la figura anterior, se despliega una ventana donde hay 4 posibilidades de elección de cámara. En este caso, se selecciona la que está marcada en rojo denominada como */camera/rgb/image_ra/compressed*.

Una vez seleccionada la cámara nos aparecerá en pantalla aquello que el robot tiene en frente suyo. A continuación, se proporciona una imagen para ver el resultado.

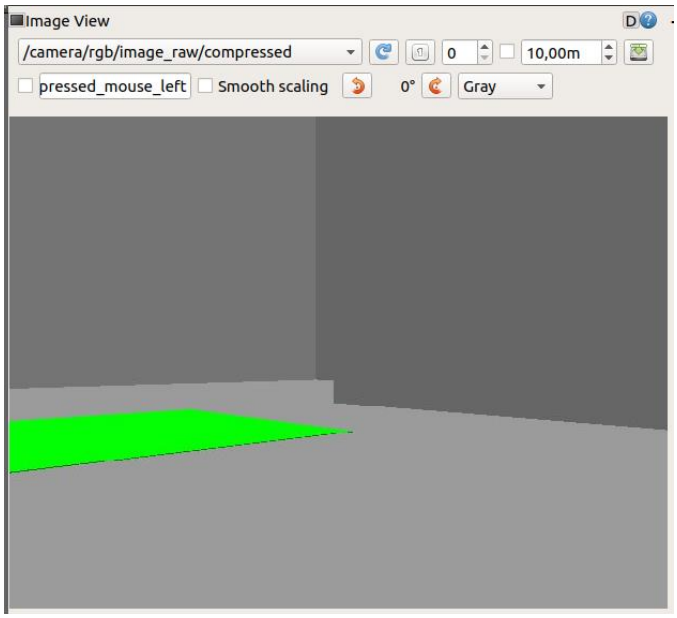


Figura 7.28 Visualización entorno a través de cámara

Fuente: Elaboración propia

Para esta ocasión, se ha activado el sensor LDS para que se observe en todo momento como actúa y donde tiene el rango de visión el TurtleBot3 en el entorno 3D de Gazebo. En la imagen que se ve a continuación, se puede observar como el robot tiene localizado todo lo que le envuelve, es decir, todo lo que es capaz de ver en el lugar donde está posicionado. Se aprecia claramente, como una vez su visión llega hasta una pared, esta se para y no visualiza a través de ella.

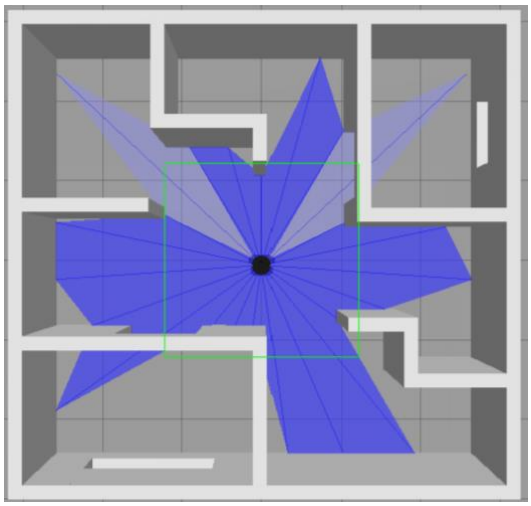

Figura 7.29 Visualización con el LDS

Fuente: Elaboración propia

Una vez se ha lanzado el programa con los comandos anteriores, así permitiendo el movimiento del robot, se muestra otra imagen desde otra posición del entorno, de esta forma poder observar que es lo que está visualizando el Waffle Pi en unas nuevas coordenadas.

En este caso, como ya se ha lanzado el programa, en el mapa se encuentra el objetivo, y como se observa en la figura 7.30, el robot se encuentra en una nueva posición visualizando un nuevo objetivo.

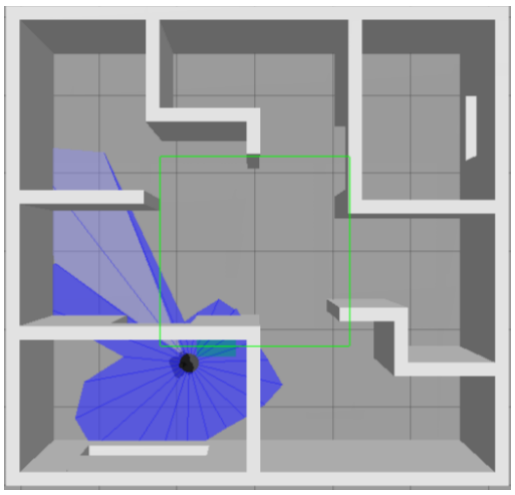

Figura 7.30 Visualización con el LDS desde otra posición

Fuente: Elaboración propia

Como ya se ha comentado anteriormente, es un proceso de aprendizaje lento y dependiendo del entorno puede tener más dificultades.

Para que se pueda interpretar y demostrar que el robot puede acceder a cualquier zona del mapa que le rodea, se muestra una imagen nueva de la posición del robot, en este caso, en el interior de una de las habitaciones.

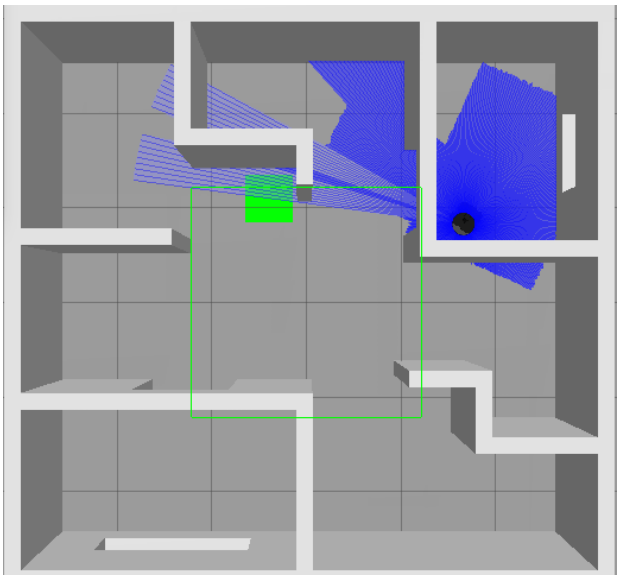

Figura 7.31 TurtleBot3 en una de las habitaciones

Fuente: Elaboración propia

A continuación, se van a mostrar varias etapas del proceso de aprendizaje de este entorno.

Primera etapa

Se establece un primer objetivo justamente debajo de la posición inicial del robot, para que éste ya consiga su primer objetivo, y ya tenga una recompensa positiva al inicio. No obstante, al iniciar el programa es poco probable que el robot alcance demasiados objetivos, es más, realizando el estudio se observa que, en la mayoría de los intentos, hasta que no lleva una gran cantidad de episodios, el robot no deja de colisionar con el entorno que le rodea.

Se aportan a continuación unas gráficas que representan las recompensas que va obteniendo y el Q-value obtenido hasta el momento.

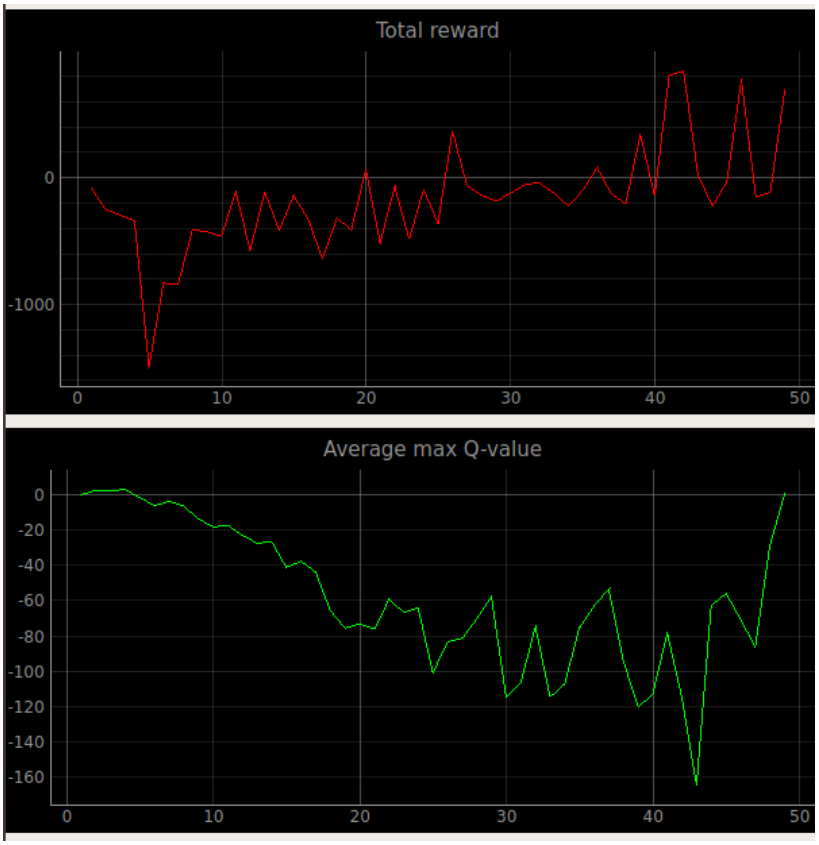


Figura 7.32 Recompensas y Q-value después de 50 episodios

Fuente: Elaboración propia

Como se observa en la figura anterior en la gráfica el Q-value se puede entender que el robot inicialmente está tomando acciones que no debería de estar realizando, esto no es nada malo, al contrario, en sus primeros pasos en el entorno está aprendiendo a saber que es lo correcto y lo incorrecto, esto provoca que al inicio tome acciones erróneas. Por otro lado, en la gráfica de las recompensas se observa como poco a poco y de vez en cuando el robot va consiguiendo los objetivos que van surgiendo en el entorno, al inicio, el robot no deja de recibir recompensas negativas debido a las colisiones que está realizando y las malas decisiones que está tomando, pero en algunos episodios se observa como las recompensas van aumentando debido a que va logrando alcanzar su propósito.

Segunda etapa

En esta segunda etapa se puede apreciar ya un conocimiento mejorado respecto la primera etapa del entorno del laberinto.

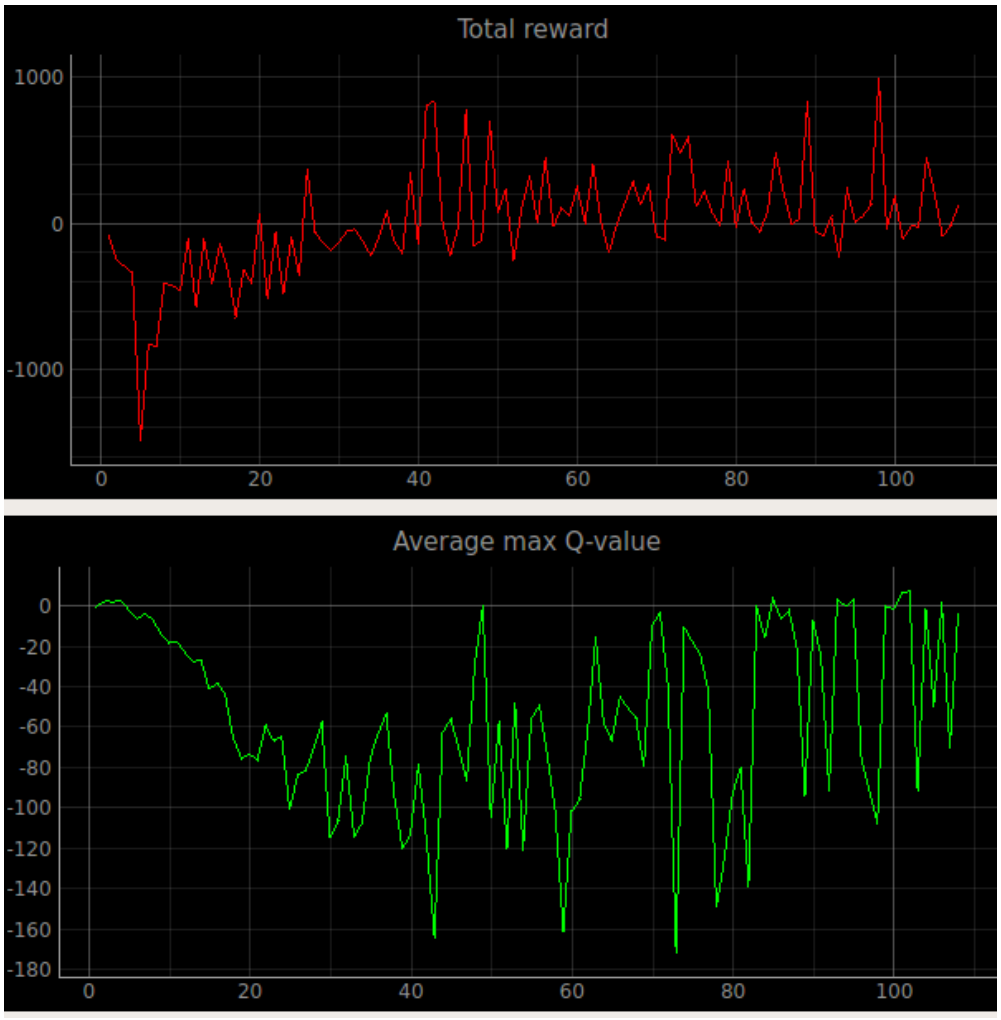


Figura 7.32 Recompensas y Q-value después de 100 episodios

Fuente: Elaboración propia

Como se observa en la figura 7.32, a partir del episodio 40 comienza a obtener de manera más constante recompensas positivas, ha pasado de tener una recompensa negativa de 1000 en el episodio 10 aproximadamente a una recompensa de 1000 cerca del episodio número 100. Esto indica que el robot está alcanzo cada vez más objetivos hasta que llegue a un punto donde deje de colisionar.

Por otro lado, el Q-value continúa siendo mayoritariamente negativo, pero cada vez está más cerca de obtener un Q-value positivo, esto quiere decir que el robot ya está empezando a tomar más decisiones y acciones correctas respecto un inicio.

Si se realiza una comparativa entre las gráficas de las etapas del entorno de la plaza y del laberinto, se aprecia claramente como la dificultad de los obstáculos que hay en el entorno del laberinto provoca un retraso del aprendizaje. Por ejemplo, mientras que la recompensa en la figura 7.16 comienza a tener puntuaciones positivas consecutivamente a partir del episodio 30, en la figura 7.32 hasta que no llega a la etapa 40 no comienza a estabilizarse en el rango positivo de recompensas. Y lo mismo sucede para el Q-value, mientras se observa como en la figura 7.13 durante los primeros 50 episodios, este valor ya da rangos positivos, por otro lado, en la figura 7.33, durante estos 50 episodios el Q-value no llega al rango positivo.

# 8. Puesta en marcha del TurtleBot3

En este apartado se explica todos los pasos que se han tenido que realizar para poder lograr la conexión entre el TurtleBot3, y el ordenador que se está utilizando para este proyecto.

Se quiere hacer una conexión vía *ssh* entre el robot y el ordenador. Lo que permite este tipo de conexión es la conectividad a un servidor remoto, haciendo uso de una comunicación cifrada mediante el puerto 22. Para realizar esto, se ha de conocer el sistema operativo de la Raspberry Pi 3 que contiene el robot, el nombre con el cual se menciona al robot y la contraseña pertinente para poder ejecutar cambios o conexiones.

Es muy importante conocer el sistema operativo que contiene la Raspberry Pi 3 ya que puede ser diferente al sistema operativo con el cual se está trabajando desde el ordenador. Como ya se comentó en apartados anteriores se está trabajando con la versión Ubuntu 18.04.

Para conocer el sistema operativo inicial del robot, se conecta la Raspberry Pi 3 a una pantalla vía HDMI. Una vez arranca en la pantalla, se observa como el sistema operativo que tenía el robot antes de comenzar este proyecto no arranca, se queda en el menú de carga de la versión Ubuntu 16.04. Al no arrancar, no se puede indagar en la información que contiene el robot y no se puede avanzar. Para solucionar esto, se retira la tarjeta microSD de la Raspberry Pi 3 y se utiliza un lector de SD para conectarlo al ordenador. Lo que se pretende en este paso, es insertar la versión Ubuntu 18.04 para que sea la misma con la que se trabaja con el sistema del ordenador. Si se intentara trabajar con versiones distintas para el PC y para el Waffle Pi no habría posibilidad de conectividad entre ellas, por este motivo, se realizan los pasos que se están describiendo.

Se utiliza el programa denominado Rufus. Esta aplicación permite crear soportes USB de arranque, suele ser útil en aquellos casos que se necesite crear medios de instalación USB a partir de ISOs arrancables, en el caso de este proyecto, la ISO de Ubuntu 18.04.LTS. En la siguiente imagen se observa como se ha utilizado para lograr posteriormente arrancar con la ISO deseada.

Se ha creado un USB bootable con la ISO de Ubuntu 18.04. Un USB bootable es una capacidad de memoria en formato USB que sirve para permitir arrancar, en este caso, la Raspberry Pi 3 desde el USB.

Se inserta de vuelta la microSD a las Raspberry Pi 3, y ésta se conecta de nuevo a la pantalla con la que se trabaja. Posteriormente, se enciende la Raspberry Pi 3, una vez encendida, se ha modificado el orden de arranque para que el USB con la ISO de Ubuntu sea lo primero y deje hacer la instalación deseada para así finalmente poder entrar al robot y continuar con las instalaciones y pasos pertinentes.

En la pantalla, sale el sistema operativo nuevo correspondiente a la versión Ubuntu 18.04. Se realiza la instalación necesaria del Ubuntu 18.04, y una vez finalizada, se le aplica un nombre para hacer referencia al robot, en este caso será el siguiente:

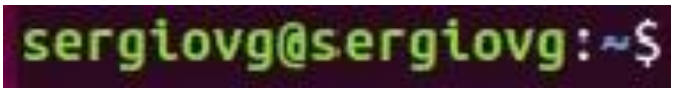


Figura 8.1 Nombre del robot

Fuente: Elaboración propia

Ahora cada vez que se quiera manipular el robot mediante cualquier tipo de conexión, se deberá de escribir el usuario de éste para poder manipularlo.

Una vez completa la instalación anterior, se pasa a conectar mediante red Wi-fi el ordenador y el TurtleBot3. Para que la conexión entre ordenador y TurtleBot3 sea posible, estos dos dispositivos deben de estar conectados a la misma red. Para lograr esto, se debe de configurar las redes de ambos dispositivos para que coincidan, una vez logrado esto se puede seguir avanzando.

Cuando ya están conectados a la misma red, se tienen que hacer varias modificaciones en el archivo *bashrc* tanto del robot como del ordenador. Primero, se va a observar los cambios realizados en el archivo del ordenador.

Para la configuración de la IP del PC se ha de realizar lo siguiente:

```
1. $ nano ~/.bashrc
```

Inicialmente, en el archivo *bashrc*, se encuentra que en el lugar de las conexiones indica *localhost*, tal y como se observa en la siguiente figura.


Figura 8.2 Configuración IP inicial del PC

Fuente: Elaboración propia

Una vez realizado este paso, se ha de cambiar donde está escrito *localhost* por la IP que está utilizando el PC en ese momento, y con la cual se va a trabajar. En este caso, se observa en la figura siguiente:

```
export ROS_MASTER_URI=http://192.168.1.146:11311
export ROS_HOSTNAME=192.168.1.146
source ~/catkin_ws/devel/setup.bash
export TURTLEBOT3_MODEL=waffle_pi
export LDS_MODEL=LDS-01
```

Figura 8.3 Configuración IP del PC

Fuente: Elaboración propia

En este caso, para la configuración del PC, el ROS_MASTER_URI y el ROS_HOSTNAME van a tener la misma dirección IP, esta dirección IP es la del PC utilizado.

Para guardar el proceso anterior, y que las IP queden configuradas, se lanza el siguiente comando:

```
1. $ source ~/.bashrc
```

Una vez configurado el archivo *bashrc* del PC, se configura el archivo que pertenece al TurtleBot3. Para este paso se realiza el mismo proceso que el anterior, se abre el archivo *bashrc*, se escribe la IP correspondiente y se guarda el archivo para que quede bien configurado. En este caso, la IP para el ROS_MASTER y la del ROS_HOSTNAME no van a ser la misma.
En este caso, inicialmente, la configuración indica lo siguiente:


Figura 8.4 Configuración IP inicial del Waffle pi

Fuente: Elaboración propia

Se ha de cambiar donde pone *localhost* por las IP correspondientes, tal y como se puede observar en la siguiente figura:

```
export ROS_MASTER_URI=http://192.168.1.146:11311
export ROS_HOSTNAME=192.168.1.54
source /opt/ros/melodic/setup.bash
source ~/catkin_ws/devel/setup.bash
export LDS_MODEL=LDS-01
```

Figura 8.5 Configuración IP del TurtleBot3

Fuente: Elaboración propia

Como se observa en la figura anterior, en el apartado del ROS_MASTER_URI se escribe la IP del PC. Por otro lado, en el apartado del ROS_HOSTNAME se escribe la IP del Waflle Pi.

Estos cambios que se han realizado son imprescindibles para la conexión entre robot y ordenador, sin ellos seria imposible que hubiese una conexión y una manipulación del TurtleBot3.

Para saber las IP de ambos aparatos electrónicos se ha de realizar el siguiente comando, dependiendo de donde se vaya a utilizar el robot el Wi-fi cambia, esto quiere decir que las IP dependiendo del sitio de donde se vaya a utilizar se deberán de ir cambiando.

1. $ ip -a

Cuando ya se tienen las IP bien configuradas en ambos casos, se debe de hacer la conexión mediante *ssh* como ya se comentó anteriormente, para poder manipular el robot desde la computadora.

Para poder realizar la conexión deseada, se ha de instalar el servicio de ssh en ambos lugares. Para ello, se ejecuta el comando que se observa a continuación:

1. $ sudo apt install openssh-server

Finalmente, cuando ya se tiene esta conexión se han de realizar las diferentes instalaciones de los paquetes necesarios en la Raspberry Pi 3 para poder trabajar con el robot sin ningún problema. Estos pasos de instalación se pueden encontrar en el apartado de anexos de este proyecto.

# 9. Navegación real por entorno desconocido

## 9.1 Navegación autónoma en entorno real

En este apartado se observa al TurtleBot3 Waffle Pi desplazarse por un entorno totalmente desconocido para él. Va a realizar un reconocimiento del entorno que le rodea, del cual no tiene ningún conocimiento previo, así siendo completamente desconocido para el robot.

Se va a hacer uso del RViz para tener visualización del progreso del mapeado que está alcanzando el TurtleBot3.

Para lograr lo comentado en el primer párrafo de este apartado del proyecto, se deben de seguir las siguientes instrucciones:

- Primero de todo, se abre una terminal en el ordenador, donde se realizará lo siguiente:
  1.$ roscore

  Esta terminal pertenece al entorno virtual del ordenador que se esta utilizando.

- Posteriormente, se abre una nueva terminal desde el PC como en el anterior punto, y dentro de ésta, se hace una conexión vía ssh. Se escribe lo siguiente:
  1.$ ssh sergiovg@sergiovg.local

  Como ya se comentó anteriormente, se realiza una conexión vía ssh. En este caso, *sergiovg* es el nombre del waffle pi que se ha otorgado en los pasos de puesta en marcha del robot. Por último, en vez de escribir la IP, se ha decidido escribir *sergiovg.local,* así permitiendo que se utilice la IP que está utilizando el robot sin la necesidad de tener que escribirla, esto último descrito es bastante útil, ya que el robot depende del sitio de donde esté conectado recibe una señal y un tipo de dirección de IP diferente, entonces escribir el nombre del robot y posteriormente escribir *.local* facilita este uso.

Una vez realizados los pasos anteriores, se tiene todo preparado para hacer uso del robot. Pero antes de empezar a que éste realice los objetivos deseados, se va a lanzar el programa RViz en el PC para ver de una forma más detallada el TurtleBot3 y el entorno que le rodea. Para ello, se realizarán los siguientes comandos:

- Se abre una nueva terminal y se escribe lo siguiente:

1. $ roslaunch turtlebot3_bringup turtlebot3_robot.launch

El lanzamiento de este paquete ya pertenece al entorno del robot. Para verlo de una forma más clara, se proporciona una imagen de como se ha realizado:

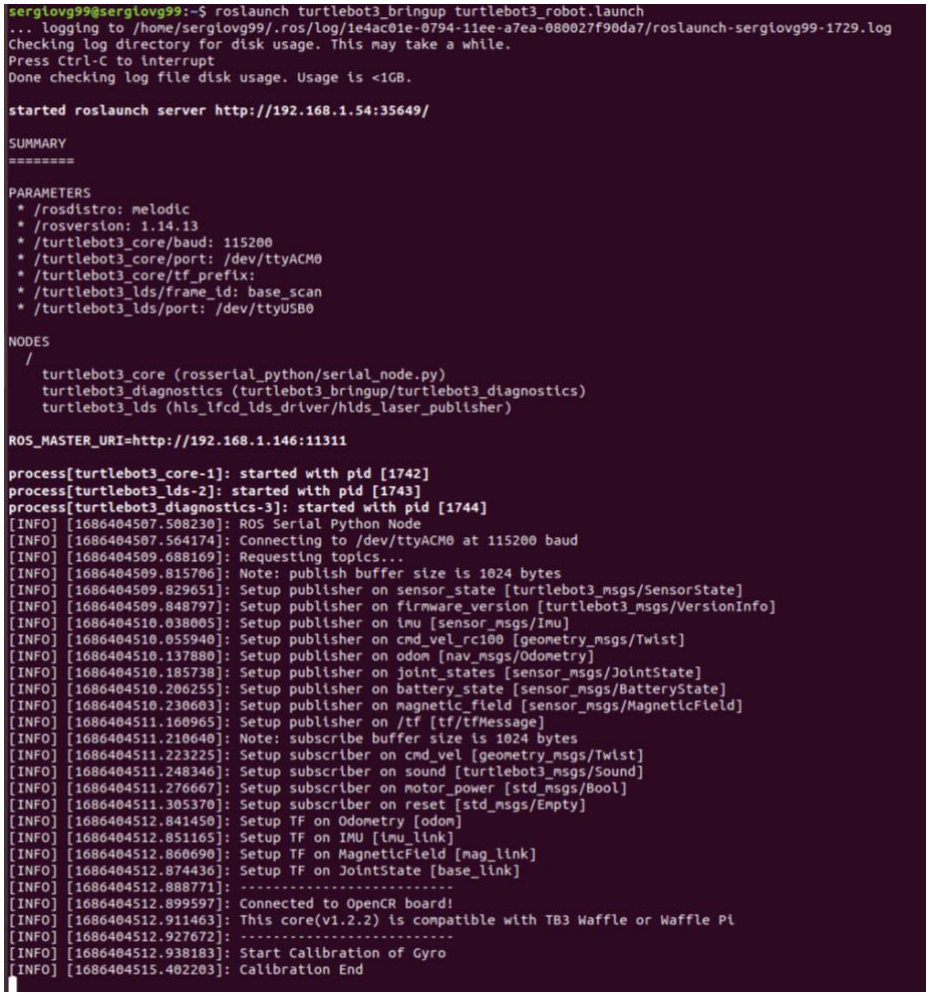


Figura 9.1 Activación del robot

Fuente: Elaboración propia

El sensor LDS del TurtleBot3 ya comienza a girar, esto quiere decir que está todo bien conectado y listo para su uso.

- Se abre otra terminal desde el PC y se escribe el modelo del TurtleBot3 que se va a utilizar. Posteriormente, se lanza el SLAM, donde se abre la aplicación RVIZ donde se va a poder observar de una forma virtual el entorno por el cual el robot real va a ir descubriendo.

  1. $ export TURTLEBOT3_MODEL=waffle_pi
  2. $ roslaunch turtlebot3_slam turtlebot3_slam.launch

Antes de seguir avanzando, se va a enseñar una imagen del primer instante de donde está situado el robot, y que es lo que puede observar a su alrededor desde la posición de donde estás, gracias al sensor LDS que contiene.

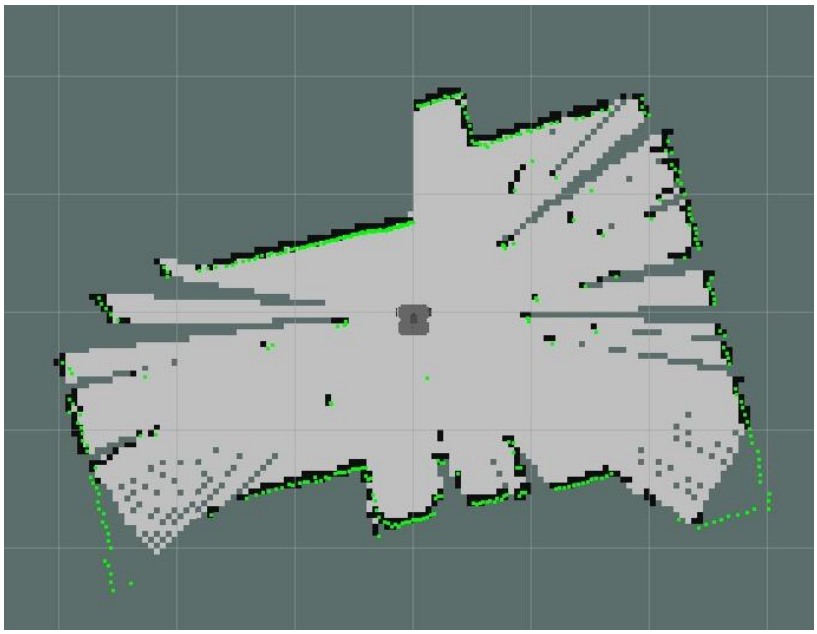

Figura 9.2 Inicio entorno desconocido

Fuente: Elaboración propia

Una vez se tienen los puntos explicados anteriormente en este apartado, primero de todo, para tener un buen inicio de contacto con el TurtleBot3, se va a utilizar el teclado para ver si hay algún problema de movimiento o de sensores del robot. Para manipular el Waffle Pi desde el teclado del ordenador, se abre una nueva terminal y en esta misma se va a realizar lo siguiente:

- Primero, se lanza el modelo con el cuál se va a trabajar.
  1.$ export TURTLEBOT3_MODEL=waffle_pi
- Posteriormente, se lanza el comando para que se pueda manipular el robot desde el teclado.
  2.$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

Una vez se tiene todo ya conectado entre sí, y con un entorno virtual para visualizar como el robot va descubriendo el entorno que le envuelve, y viendo que el robot no tiene ningún problema de desplazarse hacia atrás, ni hacia delante, ni hacia los lados, ya se puede lanzar el

programa para que el TurtleBot3 se desplace de forma autónoma sin hacer uso del teclado, únicamente mediante un código programado.

- Se abre otra nueva terminal desde el PC, donde se le lanza el programa deseado para que el robot navegue de forma autónoma. El código utilizado para esta ocasión se puede observar en la parte de los anexos de este proyecto.

Al cabo de un rato, el robot ya ha ido descubriendo partes del entorno que le rodean. Si se observa con detención la figura 9.3, se puede ver los límites de la habitación en la que se está trabajando, las patas de las mesas y las sillas, y más objetos que hay en el entorno por el cual el TurtleBot3 está navegando.

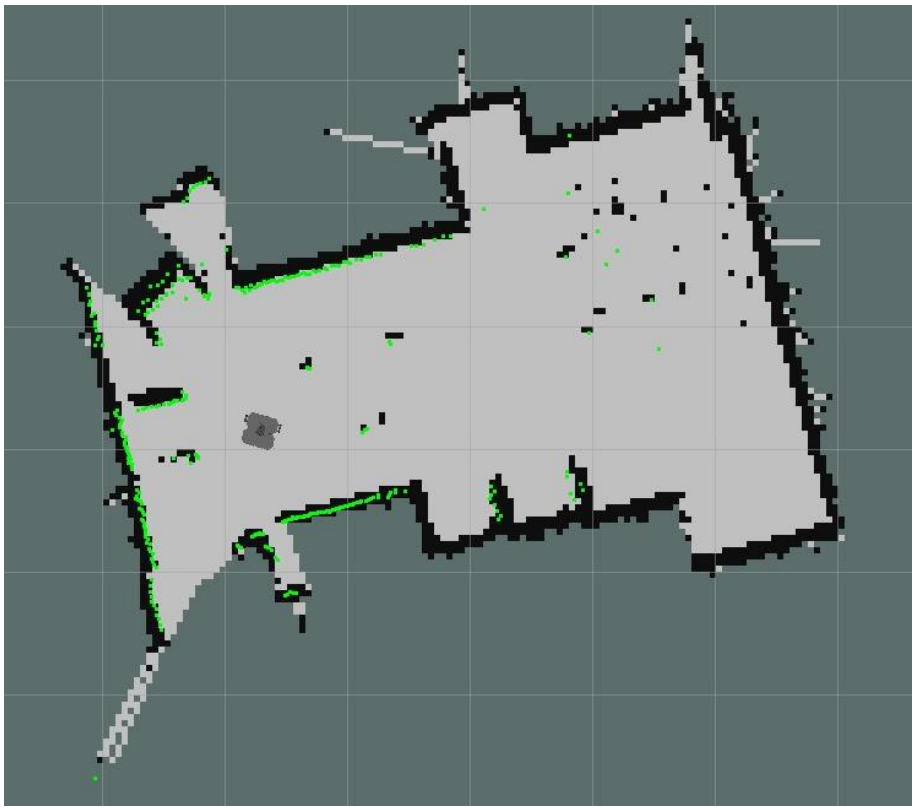

Figura 9.3 Entorno final desconocido

Fuente: Elaboración propia

Para finalizar, se guarda el mapeado final que ha terminado recorriendo el Waffle Pi. Se abre una nueva terminal y se realiza el siguiente comando:

1. $ rosrun map_server map_saver -f ~/map

Una vez se ha realizado el comando anterior, se puede ir a los archivos de la máquina virtual donde se está trabajando y se tendrá guardado la siguiente figura que se observa a continuación.

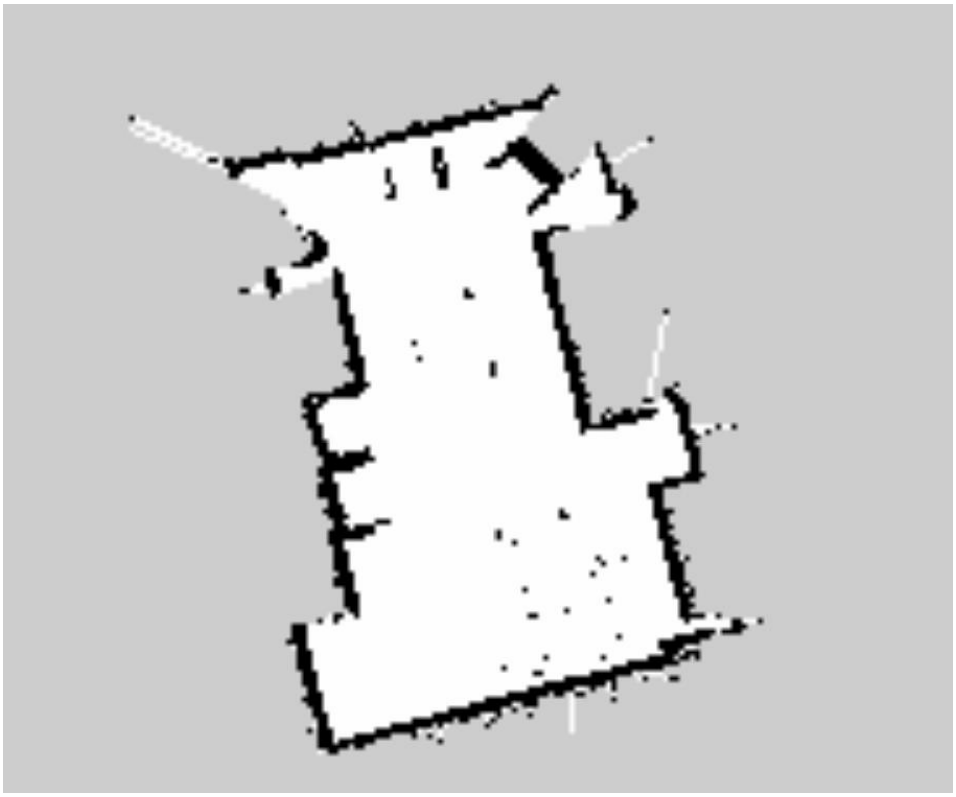

Figura 9.4 Mapa final guardado

Fuente: Elaboración propia

Ya se tiene logrado el objetivo de reconocimiento de un entorno, totalmente desconocido navegando por él de forma autónoma.

## 9.2 Navegación autónoma con redes neuronales recreando el mapa del laberinto

El código utilizado para el entrenamiento con redes neuronales en el entorno 3D Gazebo se puede utilizar en un entorno real siempre y cuando el robot esté completamente entrenado. Además, dando lugar a tener que recrear el escenario de la simulación en un entorno real, ya que el Waffle Pi detecta los obstáculos del entorno simulado y no los que pueden llegar a haber en el lugar donde se puede desplazar el robot real.

Por otro lado, una vez estén realizadas todas las conexiones e instalaciones de los paquetes necesarios, el Waffle Pi puede empezar a trabajar en un entorno real.

Cuando ya se ha entrenado el modelo en el entorno 3D Gazebo, como se ha visto en el apartado 7.2.2 y 7.2.3 de este proyecto, poniendo de ejemplo diferentes mapas, se puede lanzar el robot entrenado.

Cuando ya se han activado los sensores y se ha puesto en marcha el funcionamiento del robot, al poner en marcha el programa el robot no debe de colisionar en ningún momento con ningún obstáculo que le rodea, esto depende de las dimensiones. No obstante, el Waffle Pi tiene un tamaño no muy deseable para desplazarse en entornos reducidos.

El mapa del laberinto fue creado en unas dimensiones específicas para poder recrearlo sin ningún problema en la sala donde se iba a trabajar. Es decir, se tomaron las medidas necesarias en el entorno real para implementarlas posteriormente en el entorno 3D Gazebo, y trabajar en este simulador.

Para recordar como era el mapa del laberinto y hacer una comparativa, se presenta la imagen del entorno 3D y del entorno real a continuación.

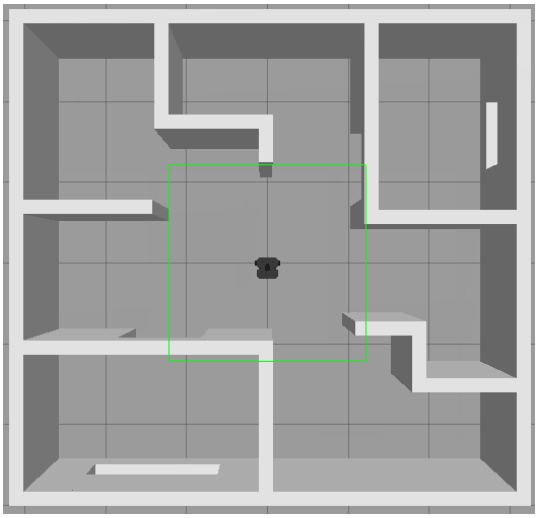

Figura 9.5 Mapa entorno 3D laberinto

Fuente: Elaboración propia

Figura 9.6 Mapa entorno real laberinto

Fuente: Elaboración propia

Como se puede observar en la figura 9.6, se ha recreado el mapa exactamente igual que en el entorno 3D, respetando todas las medidas de los espacios que contiene este mapa creado.

Finalmente, una vez se haya lanzado el programa y el modelo esté entrenado, el robot se desplaza por el mapa de forma autónoma.

## 9.3 Navegación autónoma con redes neuronales en cualquier entorno real

En este apartado se ha realizado un código para que el Turtlebot3 Waffle Pi sea capaz de desplazarse mediante redes neuronales en un entorno real, sin tener que depender de las simulaciones en Gazebo como en el caso anterior. Este código se puede observar en el apartado de anexos de este proyecto.

En este código se trabaja mediante etapas de entrenamiento, esto quiere decir que, el robot en un inicio no conocerá el entorno que le rodea, provocando que haya colisiones. Una vez, el robot se haya entrenado en dicho entorno después de horas, el Waffle Pi evitará los obstáculos que hay a su alrededor.

En este caso, se han utilizado menos capas de neuronas, menos hiperparámetros, y menos acciones. No obstante, se trabaja con el método de epsilon-greedy, ya explicado anteriormente.

### 9.3.1 Redes neuronales

Respecto a las redes neuronales utilizadas en este código, se dividen de la siguiente manera:

- Una capa de entrada, esta trabaja con el valor del *input_size* definido en el código, el valor que haya es la cantidad de neuronas de entrada que utiliza. En este caso, se trabaja con la función de activación relu.
- Una capa densa oculta con un valor de 64 neuronas, definida en *hidden_size*. En este caso, se trabaja con la función de activación relu.
- Y una capa de salida, con 3 opciones de salida: izquierda, derecha y adelante.

A continuación, se observa el esquema de las redes neuronales utilizadas:

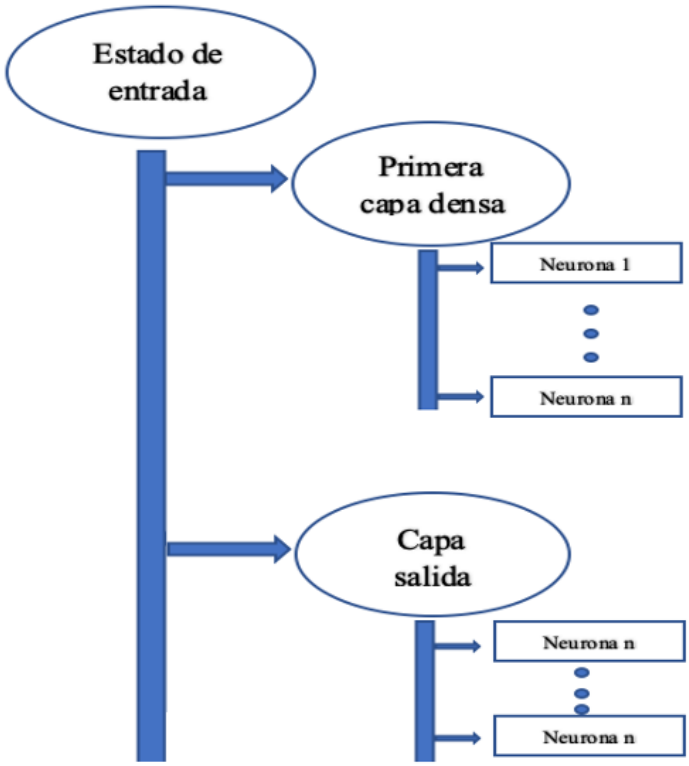


Figura 9.7 Estructura redes neuronales

Fuente: Elaboración propia

### 9.3.2 Hiperparámetros

Como ya se ha comentado en el apartado anterior, se utiliza el método Epsilon-greedy para la realización de este proyecto. Al utilizar este método, se hace uso de un seguido de parámetros, en este caso denominados hiperparámetros.

Ya que después de realizarse un estudio, y elegir este método para trabajar, es necesario implementar los hiperparámetros que se pueden observar en la tabla siguiente, con sus respectivas definiciones:

| Hiperparámetros | Definición de los hiperparámetros |
|---|---|
| Tasa de aprendizaje | Establece la tasa de aprendizaje utilizada en la optimización para actualizar los pesos de la red neuronal. |
| Épsilon | Valor inicial de la exploración (epsilon greedy). La exploración es la estrategia de tomar acciones aleatorias con el fin de descubrir nuevos conocimientos. |
| Tamaño del lote | Indica el tamaño de los lotes de muestras utilizadas para el entrenamiento de la red neuronal. Es decir, aprendizaje en lotes. |
| Épocas | Indica la cantidad de épocas de entrenamiento, es decir, indica las veces que el robot recorre los datos que recopila del entrenamiento. |

Tabla 9.1 Tabla hiperparámetros segundo código

Una vez se ha explicado las capas y los hiperparámetros que utiliza el código, y sabiendo que, inicialmente necesita un período de entrenamiento para su aprendizaje, se le facilita el aprendizaje al lanzar en el RViz, el mapa guardado de la figura 9.3. Aun así, el TurtleBot3 necesita dedicarle un tiempo considerable a navegar en dicho entorno, base a las redes neuronales.

# 10. Análisis de viabilidad

## 10.1 Viabilidad técnica

La universidad conocida como TecnoCampus dispone de dos modelos del TurtleBot3, estos modelos son el Burger y el Waffle Pi.

Para realizar el proyecto se ha decidido hacer uso del TurtleBot3 Waffle Pi, debido a que proporciona unas características y unos complementos más convenientes para el objetivo de este proyecto.

Primero de todo, se ha instalado el Oracle VM VirtualBox, una vez incorporado en el ordenador, se procede a hacer la instalación de Linux Ubuntu. Esta versión de Ubuntu es la versión 18.04.LTS. Una vez se tiene preparado, se ha de instalar y preparar el entorno ROS, en este caso, como se ha comentado en el apartado de la selección de la alternativa óptima se trabaja con ROS Melodic, y los paquetes disponibles para poder hacer la puesta en marcha del TurtleBot3 Waffle Pi.

Además, se han debido de hacer un seguido de pasos y de conexiones para poder manipular el robot en un entorno real, ya que éste desde un inicio, el contenido que tenía en su interior no era válido para la realización de este proyecto.

A lo que envuelve las simulaciones, se han elaborado un seguido de códigos y mapas para la elaboración y el estudio del aprendizaje automático, haciendo uso de entornos 3D como Gazebo, y entornos correspondientes a las redes neuronales, como son el TensorFlow y el Keras.

Es importante mencionar que el entorno ROS proporciona más flexibilidad a la hora de trabajar con módulos complejos e interacciones con el TurtleBo3 Waffle Pi.

## 10.2 Viabilidad medioambiental

Este proyecto es de carácter educativo, este hecho implica que no provoque un gran impacto medioambiental, todo lo contrario, a que si fuese un proyecto para la planta de producción de una fábrica.

El robot TurtleBot3 va alimentado por baterías recargables, lo que provoca un impacto minúsculo para el medioambiente. Estas baterías posteriormente se depositan en un punto de recogida para su posterior reciclaje.

El uso del TurtleBot3 no implica un impacto medioambiental relevante, como para hacer un análisis profundo, más bien es una acción común que puede realizar cualquier persona en su día a día.

## 10.3 Viabilidad económica

Debido a que no se trata de un proyecto orientado a un producto, se ha desarrollado la viabilidad económica desde un punto de vista diferente. Ya que es un proyecto sobre el TurtleBot3, se calculan los costes de materiales necesarios para poder realizar el proyecto.

A lo largo del proyecto, ha ido aumentando la lista de los materiales utilizados, debido a que se han ido comprando aquellos materiales necesarios para poner en funcionamiento el robot Waffle Pi.

| Material | Coste |
|---|---|
| TurtleBot 3 Waffle Pi | 1.509 € |
| Portátil Gaming ASUS ROG G14 GA401QM-K2039T, Ryzen 7, 16 GB, 1TB SSD, GeForce RTX 3060 6GB, 14'', W10 | 1.899 € |
| Webcam Logitech C270 | 27,99 € |
| Integral Lector de Tarjetas Micro SD, USB 3.1 USB 3.0, Tarjetas de Memoria Micro SD, microSDHC, microSDXC, Adaptador de Tarjetas de Memoria USB3.0 | 6,99 € |
| Hub USB 3.0 de 4 Puertos USB 3.0 HUB, OBERSTER Adaptador USB 3.0 | 12,99 € |
| Amazon Basics - Cable micro-HDMI a HDMI | 7,86 € |
| Tarjeta microSDXC 128 GB | 17,99 € |

Tabla 10.1 Coste de inversión

Como se puede observar en la tabla 10.1 todos los costes de inversión son de materiales, esto es debido a que no se trata de un proyecto el cual sea necesario una maquinaria para fabricar un producto.

# 11. Análisis de perspectiva de género

La programación y la robótica son para todas aquellas personas amantes de estas tecnologías, dicho de otra manera, este tipo de tecnologías es ciega al género, para realizar este tipo de tareas únicamente se necesita tiempo, constancia y una buena actitud para lograr conseguir ese objetivo propuesto y deseado.

Dado lo que la naturaleza de este proyecto requiere, se declara neutral en cuanto a la perspectiva de género, debido a la dificultad simétrica que implica realizar la programación tanto para hombres como mujeres.

En este proyecto se asegura que no existe ningún tipo de efectos diferenciados para hombres y mujeres a consecuencia de la realización de este trabajo.

# 12. Planificación

Se ha elaborado una planificación del proyecto donde aparecen todas las actividades que se han ido realizando.

Inicialmente, se realizó una planificación inicial con unas pautas a seguir, a medida que el proyecto iba avanzando algunas actividades se estaban alargando debido a sus dificultades o a problemas e incidencias varias tanto por tema de software como de hardware.

En la tabla 12.1, se observan las tareas principales definidas desde un inicio para este proyecto con sus respectivas actividades que se pretendían desde un principio.

Inicialmente, se habían elaborado unas pautas a seguir durante toda la realización del proyecto. En la siguiente tabla se observa como estaban distribuidas las actividades, con sus respectivas prelaciones, duraciones, y sus fechas de inicio y final. Además, si se interpreta la tabla, se observa como a partir del apartado de prelaciones se llega a la conclusión de que ha resultado ser una planificación no lineal. Es decir, durante algún período de tiempo, se tenía pensado realizar hasta varias actividades a la vez, ya que no dependían nada entre ellas para poder avanzarlas individualmente.

En el apartado 12.1 de este proyecto, se observa el diagrama de Gantt correspondiente a la planificación inicial. En este diagrama, se indica el camino crítico del proyecto, el camino crítico es aquel que está marcado en rojo en el diagrama.

| Núm. | Actividades | Prelaciones | Duración | Comienzo | Fin |
|---|---|---|---|---|---|
| 1 | Objeto del proyecto | - | 13horas | Lun 02/01/23 | Vie 06/01/23 |
| 2 | Reconocimiento de patrones y redes neuronales | 1 | 16 horas | Vie 06/01/23 | Vie 13/01/23 |
| 3 | TurtleBot3 | 1 | 24 horas | VIe 06/01/23 | Dom 15/01/23 |
| 4 | Objetivos y especificaciones técnicas | 3 | 20 horas | Dom 15/01/23 | Dom 22/01/23 |
| 5 | Generación y planteamiento de alternativas | 4 | 16 horas | Dom 22/01/23 | Sáb 28/01/23 |
| 6 | Selección de la alternativa óptima | 4,5 | 16 horas | Sáb 28/01/23 | Vie 03/02/23 |
| 7 | Ubuntu | 6 | 14 horas | Vie 03/02/23 | Jue 09/02/23 |
| 8 | ROS | 6 | 20 horas | Vie 03/02/23 | Vie 10/02/23 |
| 9 | Instalación paquetes | 7;8 | 26 horas | Vie 10/02/23 | Dom 19/02/23 |
| 10 | Introducción simulación ROS | 9 | 25 horas | Dom 19/02/23 | Mié 01/03/23 |
| 11 | Navegación autónoma del TurtleBot3 | 9,10 | 125 horas | Mié 01/03/23 | Vie 14/04/23 |
| 12 | Puesta en marcha del TurtleBot3 | 9,11 | 34 horas | Vie 14/04/23 | Jue 27/04/23 |
| 13 | Navegación real por entorno desconocido | 12 | 60 horas | Jue 27/04/23 | Vie 19/05/23 |
| 14 | Viabilidad económica | 12 | 14 horas | Jue 27/04/23 | Lun 01/05/23 |
| 15 | Viabilidad medioambiental | 13 | 8 horas | Vie 19/05/23 | Sáb 20/05/23 |
| 16 | Viabilidad económica | 13 | 21 horas | Vie 19/05/23 | Vie 26/05/23 |
| 17 | Perspectiva de género | 16 | 6 horas | Vie 26/05/23 | Sáb 27/05/23 |
| 18 | Planificación | 16 | 23 horas | Vie 26/05/23 | Vie 02/06/23 |

Tabla 12.1 Planificación inicial

Una vez se ha visto esta primera planificación, se puede observar la planificación final. En esta, no ha sido posible seguir los pasos y los tiempos que respetan la planificación inicial. A medida que las fechas han ido avanzando, se puede contemplar como algunas actividades no estaban siendo correspondidas respecto a los tiempos iniciales.

Estos contratiempos se han debido a que han surgido problemas e incidencias en algunas partes del proyecto que se han alargado más de lo que se había esperado, ya hayan sido errores con las simulaciones, problemas con el código o incidentes con el robot real.

Aún intentando predecir los tiempos en la planificación inicial, aún imaginando y haciendo una aproximación de las complicaciones que suelen haber con los códigos de programación, las máquinas virtuales y las configuraciones en un entorno real, no ha sido posible mantener esta planificación inicial.

Además, durante la realización del proyecto, se han ido adquiriendo conocimientos, dichos conocimientos han provocado un retraso inesperado ya que desde un inicio no se había tenido en cuenta que para las redes neuronales se necesitaban un seguido de códigos abiertos como son TensorFlow y Keras. Al adquirir esta nueva información, se ha realizado un estudio de ambas bibliotecas lo que ha provocado una suma de tiempo respecto al inicio del proyecto.

A continuación, en la tabla 12.2, se observa como ha quedado finalmente la planificación de este proyecto. En dicha tabla se encuentran los nombres de las actividades, que, en este caso, hay dos más respecto la planificación inicial, las prelaciones, y las fechas de inicio y final de cada una de las actividades.

En el apartado 12.2 de este proyecto, se observa el diagrama de Gantt correspondiente a la planificación final. El camino crítico es aquel que está marcado en rojo en el diagrama.

| Núm. | Actividades | Prelaciones | Duración | Comienzo | Fin |
|---|---|---|---|---|---|
| 1 | Objeto del proyecto | - | 16 horas | Lun 02/01/23 | Sáb 07/01/23 |
| 2 | Reconocimiento de patrones y redes neuronales | 1 | 16 horas | Sáb 07/01/23 | Vie 13/01/23 |
| 3 | TurtleBot3 | 1 | 24 horas | Sáb 07/01/23 | Lun 16/01/23 |
| 4 | Objetivos y especificaciones técnicas | 3 | 16 horas | Lun 16/01/23 | Dom 22/01/23 |
| 5 | Generación y planteamiento de alternativas | 4 | 14 horas | Dom 22/01/23 | Vie 27/01/23 |
| 6 | Alternativa óptima | 4,5 | 10 horas | Vie 27/01/23 | Mar 31/01/23 |
| 7 | Ubuntu | 6 | 8 horas | Mar 31/01/23 | Vie 03/02/23 |
| 8 | ROS | 6 | 18 horas | Mar 3 1/01/23 | Lun 06/02/23 |
| 9 | Instalación paquetes | 7;8 | 20 horas | Lun 06/02/23 | Lun 13/02/23 |
| 10 | TensorFlow | 9 | 12 horas | Lun 13/02/23 | Vie 17/02/23 |
| 11 | Keras | 9 | 6 horas | Lun1 3/02/23 | Vie 07/02/23 |
| 12 | Introducción simulación ROS | 9 | 30 horas | Lun 13/02/23 | Vie 24/02/23 |
| 13 | Navegación autónoma del TurtleBot3 | 9,12 | 145 horas | Vie 24/02/23 | Dom 16/04/23 |
| 14 | Puesta en marcha del TurtleBot3 | 9,13 | 40 horas | Dom 16/04/23 | Lun 01/05/23 |
| 15 | Navegación real por entorno desconocido | 14 | 70 horas | Lun 01/05/23 | Vie 26/05/23 |
| 16 | Viabilidad técnica | 14 | 14 horas | Lun 01/05/23 | Sáb 06/05/23 |
| 17 | Viabilidad medioambiental | 15 | 8 horas | Vie 26/05/23 | Lun 29/05/23 |
| 18 | Viabilidad económica | 15 | 26 horas | Vie 26/05/23 | Dom 04/06/23 |
| 19 | Perspectiva de género | 18 | 6 horas | Dom 04/06/23 | Mié 07/06/23 |
| 20 | Planificación | 18 | 26 horas | Dom 04/06/23 | Mié 14/06/23 |

Tabla 12.2 Planificación final

## 12.1 Diagrama de Gantt inicial

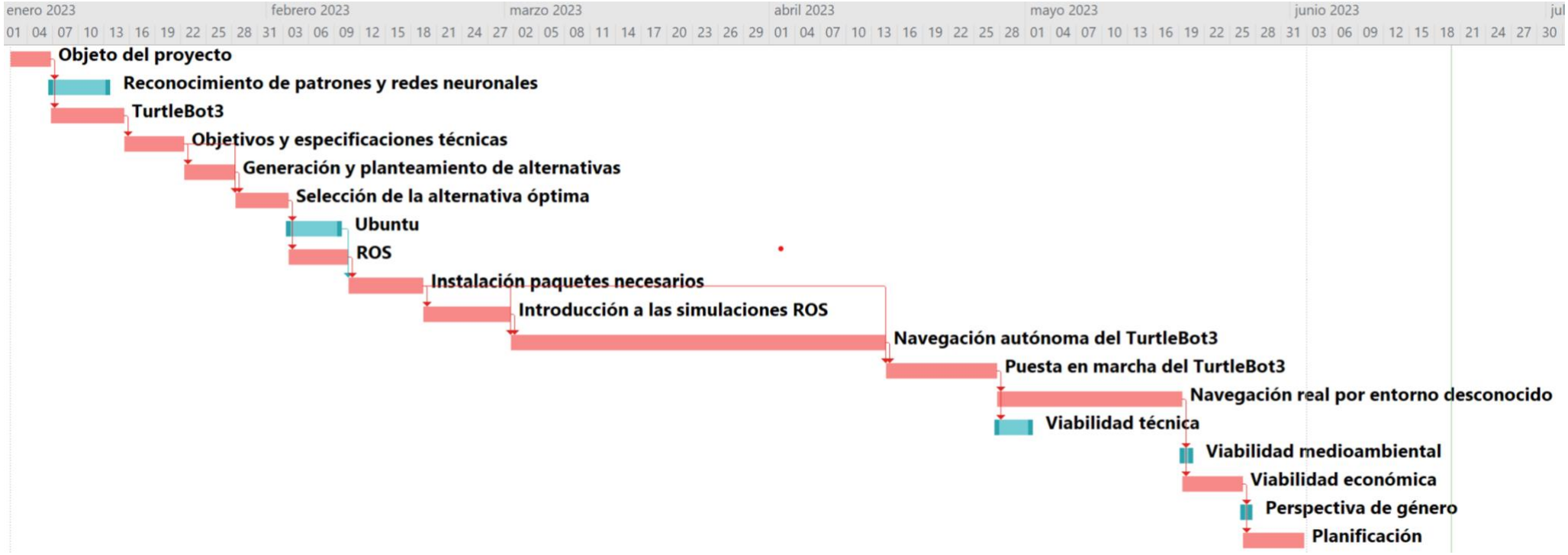


Figura 12.1 Diagrama de Gantt inicial

Fuente: Elaboración propia

## 12.2 Diagrama de Gantt final

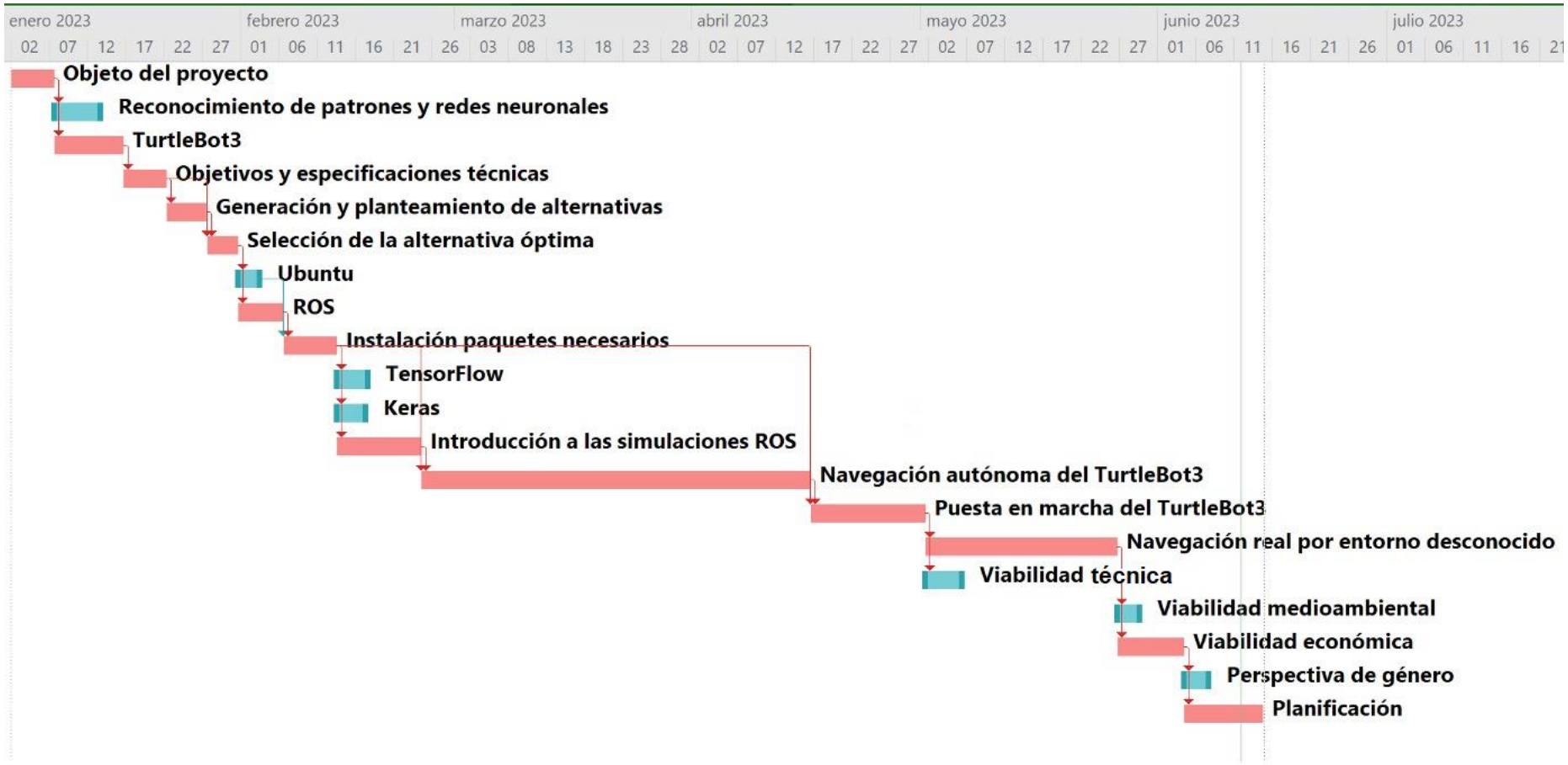


Figura 12.2 Diagrama de Gantt final

Fuente: Elaboración propia

# 13. Conclusiones

En este documento se ha explicado y se ha hecho un análisis profundo de todos aquellos conceptos relevantes para la realización y el inicio de la puesta en marcha del TurtleBot3.

Se han especificado los requerimientos técnicos y de información necesarios para la realización de este proyecto, los análisis correspondientes tanto del medioambiente como el económico. También, se proporciona un análisis detallado de los pasos a seguir durante toda la realización del proyecto hasta lograr el objetivo deseado, viendo y comprendiendo el funcionamiento del robot para que trabaje en base al aprendizaje autónomo y las redes neuronales.

Haciendo este estudio y habiendo realizado este proyecto mediante el tipo de entrenamiento que se ha comentado, se puede comprender que el TurtleBot3 requiere de bastante tiempo para lograr todo este aprendizaje autónomo comentado, no es un proceso rápido, ya que, dicho de otra manera, se le está enseñando al robot a dar sus primeros pasos en un entorno completamente desconocido para él. Para ello, el TurtleBot3 necesita tiempo y aprendizaje.

Se observa en este proyecto gracias a los dos ejemplos de entorno con los que se trabaja, como el entorno que rodea al TurtleBot3 afecta a su aprendizaje autónomo, esto quiere decir, que dependiendo de los obstáculos que haya a su alrededor, el Waffle Pi llega a tener más dificultades de aprender, así necesitando más tiempo para lograr el objetivo deseado en aquellos escenarios que contengan más dificultades para navegar.

Durante el estudio con el robot de forma física, han surgido imprevistos los cuáles han provocado un retraso en la planificación de la memoria final. El robot tenía un correcto funcionamiento, ya que se había estado trabajando y haciendo un estudio con él sin ningún problema hasta la instalación de los paquetes de TensorFlow y Keras. Una vez instalados estos paquetes, el robot cuando estaba encendido no funcionaba con ningún código, ni autónomo, ni manejándolo con el teclado del PC. Posteriormente, no dejaba lanzar el comando del *bringup*, hecho que no permitía que el sensor LDS comenzase a girar y funcionar, lo único que se permitía era la conexión vía SSH entre el PC y el robot. Tras varios intentos de posibles soluciones, y viendo que ninguna de ellas estaba siendo útiles ya que el robot no funcionaba, se decidió eliminar todo el contenido que había en el interior de la Raspberry Pi 3, y volver a hacer la instalación del Ubuntu correspondiente, y posteriormente, instalar de nuevo todos los

paquetes necesarios. Una vez realizado todo este reinicio, el robot volvió a funcionar a la perfección.

En el estudio que se ha realizado, englobando tanto el entorno virtual como el real, han surgido muchas incidencias durante su realización, en general con el sistema operativo ROS. Aún teniendo estas dificultades y utilizando un sistema operativo con el cual tenía pocos conocimientos sobre él antes de empezar este proyecto. A fecha de hoy después de haberlo realizado, me siento muy satisfecho por haber podido resolver los problemas que me han ido apareciendo, lo que ha provocado que mis conocimientos de este entorno respecto el inicio hayan aumentado.

Además, se han planificado las actividades que se han de ejecutar y el proyecto de la herramienta MS-Project generándose los diagramas de Gantt y calculando todos los costes asociados a la ejecución y la planificación. Respecto a los costes materiales, han ido aumentando según ha ido avanzando el proyecto debido a la necesidad de uso de unos materiales específicos para poder trabajar con el Waffle Pi en un entorno real

## 13.1 Líneas futuras de trabajo

Como líneas futuras de trabajo hay una ventana muy amplia, ya que el sistema operativo ROS cada vez es más conocido en el mundo de la robótica.

El entorno ROS y el robot TurtleBot3 ofrece un mundo inmenso de exploración, en este proyecto se ha indagado y se ha adquirido conocimiento de RViz, Gazebo, ROS, del TurtleBot3 Waffle pi, de la navegación autónoma, de redes neuronales, de entrenar modelos, y más ámbitos que engloban hoy en día la robótica.

Aún habiendo hecho un estudio de todo lo que se ha comentado, hay muchas más formas de trabajar con estos entornos. Una solución interesante, podría ser la incorporación de una cámara para reconocer el entorno que rodea al robot, y dependiendo de lo que detecte actúe de una manera u otra. Por ejemplo, si el robot identifica una señal de tráfico como la señal *stop*, éste debería de parar, o una señal de aparcamiento y que el robot aparcase. Y así, realizando diferentes opciones según lo que la cámara que se haya incorporado detecte. Realizar este estudio tanto en un entorno real, como con simulaciones en Gazebo, ya que es un entorno 3D bastante interesante y donde se pueden crear ambientes muy realistas.

Otra posible solución sería incorporar un sensor de voz, y mediante comandos de voz el robot actuase de una manera u otra, dependiendo de las instrucciones que el usuario le indique.

Hasta se podría realizar un estudio del cambio de la estructura del robot, que materiales se podrían utilizar en la estructura del Waffle Pi, y que estos fuesen reciclables o modulables, para así contribuir a la sostenibilidad en un proyecto como éste en un futuro.

# 14. Referencias

[1] Reconocimiento de patrones, Wikipedia. Febrero 2022: Reconocimiento de patrones - Wikipedia, la enciclopedia libre.

[2] Redes neuronales artificiales, Wikipedia. Febrero 2023: https://es.wikipedia.org/wiki/Red_neuronal_artificial.

[3] Turtlebot, 2021: https://www.ro-botica.com/Producto/ROBOTIS-TurtleBot3-Burger/

[4] TurtleBot 3 Burger. ROS Components 2016: https://www.roscomponents.com/es/robots-moviles/214-turtlebot3-burger.html#/cursos-no/turtlebot_3_burger_modelo-burger_intl_

[5] Componentes TurtleBot3 Burger. ROTATECNO 2020: https://www.rotatecno.com/producto/turtlebot-3-burger/

[6] Motor DYNAMIXEL (XL430-W250-T): https://sandorobotics.com/producto/902-0135-000/

[7] Motor DYNAMIXEL (XL430-W250-T). ROBOTIS 2023: https://emanual.robotis.com/docs/en/dxl/x/xl430-w250/

[8] Placa OpenCR 1.0. RO-BOTICA 2021: https://www.ro-botica.com/Producto/OpenCR1%7C0/

[9] Raspberry Pi 3: https://www.amazon.es/Raspberry-Pi-Modelo-Quad-Core-Cortex-A53/dp/B01CD5VC92

[10] Raspberry Pi 3. MXIElectronics.cl: https://raspberrypi.cl/raspberry-pi-3b/

[11] Sensor de distancias láser LDS-01: https://sandorobotics.com/producto/903-0258-000/

[12] TurtleBot3 Waffle Pi. ROS Components 2016: https://www.roscomponents.com/es/robots-moviles/215-turtlebot3-waffle-pi.html#/cursos-no/turtlebot3_waffle_pi_modelo-waffle_pi

[13] Componentes TurtleBot3 Waffle Pi: https://www.smartrobotworks.com/TurtleBot3-Waffle-Pi.asp

[14] Motor DYNAMIXEL (XM430-W210-T): https://sandorobotics.com/producto/902-0125-000/?_x_tr_sl=es&_x_tr_tl=ca&_x_tr_hl=ca&_x_tr_pto=sc

[15] Motor DYNAMIXEL (XM430-W210-T): https://www.trossenrobotics.com/dynamixel-xm430-w210-t.aspx

[16] Raspberry Pi cámara: https://cdn.sparkfun.com/datasheets/Dev/RaspberryPi/RPiCamMod2.pdf

[17] Raspberry Pi cámara: https://tienda.bricogeek.com/accesorios-raspberry-pi/822-camara-raspberry-pi-v2-8-megapixels.html

[18]Ubuntu: https://www.ma-no.org/es/redes/servers/que-es-apparmor-y-como-mantiene-seguro-ubuntu

[19] Tipos de software, ROS. Open Robotics 2021: http://wiki.ros.org/es

[20] RVIZ: https://github.com/ros-visualization/rviz

[21] Conceptos principales, ROS. Erle Robotics: https://erlerobotics.gitbooks.io/erlerobot/content/es/ros/ROS-concepts.html

[22] Paquetes, ROS. Pilz GmbH & Co.KG: https://www.pilz.com/es-ES/products/robotics/ros-modules.

[23] AppArmor, Ubuntu. 2022: https://es.wikipedia.org/wiki/AppArmor

[24] Documentación Gazebo: https://rych.dcc.uchile.cl/doku.php?id=documentacion:gazebo

[25] Gazebo. Generation Robots: https://www.generationrobots.com/blog/en/robotic-simulation-scenarios-with-gazebo-and-ros/

[26] TensorFlow. 2021: https://es.wikipedia.org/wiki/TensorFlow

[27] ¿Qué es TensorFlow y para que sirve?. Jon Larkin Alonso. 16/06/2022: https://www.incentro.com/es-ES/blog/que-es-tensorflowf

[28] Keras: https://datascientest.com/es/keras-la-api-de-deep-learning

[29] Robotis. 2023: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[30] Método épsilon-greedy. Analytics Lane, 2018-2023: https://www.analyticslane.com/2021/03/05/epsilon-greedy-con-decaimiento-para-un-problema-bandido-multibrazo-multi-armed-bandit/#:~:text=La%20estrategia%20Epsilon%2DGreedy%20con%20decaimiento&text=Algo%20que%20no%20es%20necesario,aumenta%20el%20número%20de%20episodios.

[31] Método Softmax. Jacar: https://jacar.es/funcion-softmax-activacion-para-la-clasificacion/#:~:text=La%20función%20SoftMax%20es%20una%20función%20de%20activación%20que%20se,en%20probabilidades%20que%20suman%20uno.