# Documentación del Proyecto: Robot Móvil Autónomo con LiDAR y SLAM

## 1. Resumen del Proyecto

Este proyecto detalla el diseño y la implementación de un sistema de software para un robot móvil autónomo. El sistema es capaz de:

* **Percibir su entorno** en 360 grados utilizando un sensor LiDAR.
* **Construir un mapa 2D** del entorno en tiempo real utilizando un algoritmo de Mapeo de Rejilla de Ocupación (una forma de SLAM).
* **Ser controlado remotamente** a través de una interfaz gráfica.
* **Planificar y simular** rutas de cobertura autónoma de forma eficiente.

El sistema se divide en dos componentes principales: un **firmware para el microcontrolador ESP32** que adquiere los datos, y una **aplicación de escritorio en Python** que actúa como el cerebro central para el procesamiento, visualización y control.

## 2. Módulos de Software en Detalle

### 2. Implementación del Sistema Maestro

El corazón del robot, o "controlador maestro", es el responsable de las tareas de alto nivel: el procesamiento del Lidar, la ejecución del algoritmo SLAM y, eventualmente, la inferencia de la red neuronal. La plataforma seleccionada para esta tarea es la **Raspberry Pi**.

**3.1. Plataforma de Cómputo: Raspberry Pi 4**

Se seleccionó una Raspberry Pi 4 Modelo B como la computadora principal del sistema. Esta elección se basa en su balance entre potencia de cómputo (CPU ARM Cortex-A72 de 64 bits), consumo energético y la disponibilidad de periféricos clave:

**Puertos USB 3.0:** Necesarios para la transferencia de datos de alta velocidad desde el sensor Lidar.

**Pines GPIO:** Utilizados para establecer la comunicación serial (UART) con el microcontrolador esclavo (ESP32).

**Capacidad de RAM:** (Menciona tu versión, ej. 4GB u 8GB) Suficiente para ejecutar un sistema operativo completo y los nodos de ROS 2.

**3.2. Sistema Operativo (OS): Ubuntu 22.04 LTS (64-bit)**

La elección del sistema operativo es crítica para la compatibilidad del software. Tras un análisis preliminar, se descartó el uso de Raspberry Pi OS de 32 bits (anteriormente "Buster"), ya que presentaba conflictos irresolubles de dependencias al intentar instalar **ROS 1 Noetic**.

Se optó por instalar **Ubuntu 22.04 LTS (Jammy Jellyfish)** en su versión de **64 bits (aarch64)**. Esta decisión estratégica ofrece tres ventajas fundamentales:

**Soporte Nativo:** Es la plataforma principal de desarrollo y soporte para ROS 2 Humble.

**Compatibilidad de Paquetes:** Permite la instalación de paquetes binarios (vía apt) directamente, eliminando la necesidad de compilar ROS desde el código fuente, un proceso que consume mucho tiempo y es propenso a errores.

**Arquitectura de 64 bits:** Es un requisito indispensable para las bibliotecas modernas de cómputo y aprendizaje automático (como TensorFlow) que se utilizarán en fases posteriores del proyecto.

**3.3. Middleware Robótico: ROS 2 Humble Hawksbill**

Como framework de software se utiliza el **Robot Operating System 2 (ROS 2)**, en su distribución **Humble Hawksbill**.

ROS 2 gestiona toda la comunicación entre los diferentes procesos de software (nodos). La elección de la versión Humble está directamente ligada al uso de Ubuntu 22.04, ya que es la versión de soporte a largo plazo (LTS) designada para este sistema operativo.

Esta pila de software (Ubuntu 22.04 + ROS 2 Humble) nos permite instalar y ejecutar de manera limpia los paquetes de software necesarios para el proyecto:

**ldlidar-stl-ros2**: El driver oficial para nuestro sensor Lidar.

**slam-toolbox**: El paquete estándar de ROS 2 para la implementación de SLAM.

## 4. Modelo de Navegación: Red Neuronal por Imitación

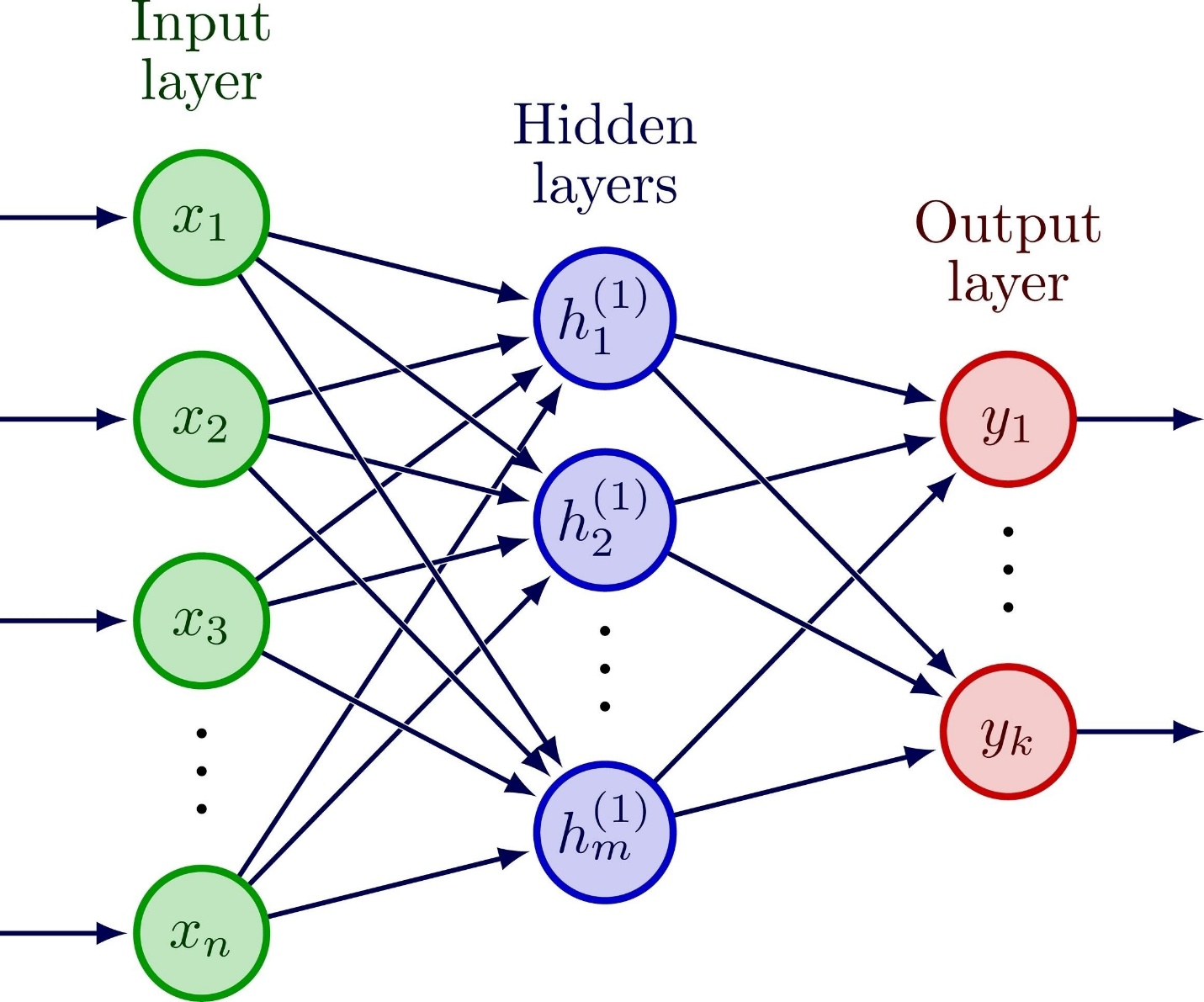
Para la generación de rutas y la evasión de obstáculos en tiempo real, este proyecto utiliza un enfoque de **Aprendizaje por Imitación (Imitation Learning)**, también conocido como **Clonación de Comportamiento (Behavioral Cloning)**.

A diferencia de los planificadores de ruta clásicos (como el algoritmo A\*), que calculan una ruta óptima usando un mapa estático, nuestro modelo de red neuronal funciona como un "cerebro" de conducción. Su objetivo es aprender a **mapear directamente una situación sensorial (lo que ve el Lidar) a una acción motora (cómo mover los motores)**.

En esencia, entrenamos a la red para que "conduzca como un humano".

**Cómo Funciona el Modelo**

El modelo de red neuronal es un sistema que toma un conjunto de números como entrada y, tras una serie de cálculos, entrega otro conjunto de números como salida.



Shutterstock

Explorar

**1. Entrada (Input): Los "Ojos" del Robot** La entrada de la red es un **vector de datos** que representa el entorno, proveniente directamente del Lidar D500.

* Tras un escaneo, el Lidar nos entrega un conjunto de mediciones (ej. 360 puntos) que indican la distancia a los obstáculos en cada ángulo.
* **Ejemplo de Entrada:** [dist\_1, dist\_2, dist\_3, ..., dist\_360]

**2. Salida (Output): Las "Manos" del Robot** La salida de la red es un vector simple que define la acción que debe tomar el chasis.

* Para un robot de tracción diferencial (dos ruedas), solo necesitamos dos valores.
* **Ejemplo de Salida:** [velocidad\_rueda\_izquierda, velocidad\_rueda\_derecha]

**3. La Red (El "Cerebro"):** Entre la entrada y la salida hay una serie de "capas ocultas" de neuronas. Se propone usar una **Red Neuronal Densa (Dense Neural Network)**, también llamada Perceptrón Multicapa (MLP).

* Cada "neurona" es una pequeña unidad matemática que pesa las entradas, suma un sesgo y aplica una función de activación (como ReLU).
* Al apilar estas capas, la red aprende a encontrar patrones complejos. Por ejemplo, puede aprender que "si las distancias [dist\_1 a dist\_40] son muy pequeñas, entonces la [velocidad\_rueda\_derecha] debe ser negativa y la [velocidad\_rueda\_izquierda] positiva para girar y evitar el obstáculo".

**Cómo se Entrena (El Dataset)**

Aquí es donde entra en juego la **Clonación de Comportamiento**. La red aprende de un dataset que **nosotros mismos debemos crear**.

**Metodología de Recolección de Datos:**

1. **Conducción Manual:** Se operará el robot manualmente usando un joystick o teclado conectado a la Raspberry Pi. Conduciremos el robot por el patio, esquivando árboles, paredes y siguiendo caminos.
2. **Registro de Datos (Data Logging):** Mientras conducimos, un script en la Pi estará "grabando" todo. Por cada instante de tiempo (ej. cada 100 milisegundos), guardará un par de datos:
   * **Dato (X):** El escaneo completo del Lidar en ese momento.
   * **Etiqueta (Y):** El comando de motor que el humano envió en ese *exacto* momento (ej. [100, 100] para avanzar, [-80, 80] para girar).
3. **Creación del Dataset:** Después de 15-20 minutos de conducción, tendremos miles de estos pares (X, Y). Este conjunto de datos es nuestro "libro de texto" para la red neuronal.
4. **Entrenamiento:** Alimentamos este dataset a un framework como TensorFlow/Keras. La red (el "estudiante") intentará predecir una salida Y\_pred basándose en la entrada X. El algoritmo de entrenamiento (optimizador) comparará la predicción Y\_pred con la etiqueta real Y (lo que hizo el humano) y ajustará los pesos de la red para reducir el error.
5. **Resultado:** Después de miles de iteraciones, la red aprende a "clonar" el comportamiento del conductor humano.

**Integración en el Sistema**

Una vez entrenado, el modelo se optimiza (usando **TensorFlow Lite**) y se carga en la Raspberry Pi. El flujo de trabajo en el robot final será:

1. El Lidar D500 envía su escaneo a la Pi (Maestro).
2. La Pi alimenta este escaneo a la Red Neuronal (modelo TFLite).
3. La Red Neuronal devuelve una predicción de motor (ej. [90, 85]).
4. La Pi formatea esta predicción en nuestro protocolo serial (ej. <M:90,85>).
5. La Pi envía este comando por UART al ESP32 (Esclavo).
6. El ESP32 recibe el comando y ajusta el PWM de los motores.

Este enfoque permite que el robot reaccione a obstáculos que ni siquiera existían cuando se creó el mapa SLAM.

**Fase 1: Preparación (Prerrequisitos)**

Antes de que el robot pueda operar solo, necesita dos cosas: un "mapa" (para SLAM) y un "cerebro" (el modelo de Red Neuronal).

**A. Fase de Mapeo (SLAM)**

1. **Acción:** Inicias los nodos de ROS 2 para el Lidar (ldlidar\_stl\_ros2) y el SLAM (slam\_toolbox).
2. **Proceso:** Conduces manualmente el robot (con un joystick o teclado) por todo el exterior.
3. **Resultado:** slam\_toolbox usa los datos del Lidar para generar un mapa 2D (map.yaml). **Este mapa se guarda.**

**B. Fase de Entrenamiento (Red Neuronal)**

1. **Acción:** Inicias un script especial de "grabación" (Logger) en la Pi.
2. **Proceso:** Vuelves a conducir manualmente el robot, esquivando obstáculos y siguiendo caminos. El script "Logger" graba miles de líneas, guardando en cada instante:
   * **Dato de Entrada (X):** El escaneo del Lidar (360 distancias).
   * **Dato de Salida (Y):** Tu comando de motor en ese momento (ej. [100, 80]).
3. **Resultado:** Obtienes un **dataset**. Usas este dataset *fuera* del robot (en tu PC) para entrenar tu modelo de Red Neuronal. El modelo entrenado (ej. model.tflite) se copia de vuelta a la Raspberry Pi.

**Fase 2: Flujo de Operación Autónoma (El Robot en Acción)**

Este es el bucle principal que se ejecuta cuando enciendes el robot para que limpie. El robot **ya tiene el modelo de Red Neuronal cargado** en la Pi.

**1. Encendido y Arranque**

* **Raspberry Pi (Maestro):** Arranca Ubuntu, inicia ROS 2 y carga tu script de navegación (que contiene la Red Neuronal).
* **ESP32 (Esclavo):** Arranca su código y se pone en modo "escucha", esperando comandos por el puerto serial.

**2. Percepción (Ojos)**

* El **Lidar D500** gira y mide las distancias.
* Envía los datos crudos por USB a la **Raspberry Pi**.
* El nodo ldlidar\_stl\_ros2 (en la Pi) procesa estos datos y los publica en el topic /scan de ROS 2.

**3. Decisión (Cerebro)**

* Tu script de navegación (en la Pi) está suscrito al topic /scan.
* Recibe el vector de distancias del Lidar (ej. [5.1, 5.0, ..., 1.2, 1.1, ...]).
* Este vector se introduce como **Entrada** a tu **Red Neuronal** (model.tflite).
* La Red Neuronal procesa los datos y genera una **Salida**: dos números que representan la velocidad de los motores (ej. [110, 130] para girar suavemente a la izquierda).

**4. Comando (Voz)**

* Tu script de navegación toma la salida [110, 130].
* La formatea en el protocolo que definimos (ej. la cadena de texto **<M:110,130>**).
* La **Raspberry Pi** envía esta cadena de texto a través de sus pines **UART (TX)**.

**5. Recepción (Oídos)**

* El pin **UART (RX)** del **ESP32** recibe la cadena <M:110,130>.
* El código del ESP32 la lee, comprueba que el formato es correcto y extrae los números 110 (izquierda) y 130 (derecha).

**6. Acción (Músculos)**

* El **ESP32** usa los valores 110 y 130 para ajustar las señales **PWM** (modulación de ancho de pulso) que envía a los *drivers* de los motores.
* Los motores giran a las velocidades especificadas, moviendo el robot.

**7. Repetición (El Bucle)**

* El Lidar completa otra vuelta, se publica un nuevo /scan y todo el ciclo (Pasos 2 al 6) se repite.
* Esto sucede aproximadamente **10 veces por segundo**, permitiendo al robot reaccionar y conducir de forma fluida.

**Resumen del Flujo de Datos**

Aquí tienes un resumen simplificado del flujo de datos en la Fase Autónoma:

**LIDAR** ➡️ [USB] ➡️ **RASPBERRY PI** ➡️ [ROS 2 Topic: /scan] ➡️ **RED NEURONAL** ➡️ [Predicción: 110,130] ➡️ [UART: "<M:110,130>"] ➡️ **ESP32** ➡️ [PWM] ➡️ **DRIVERS DE MOTOR** ➡️ **MOTORE**

## Preparacon de corrida

**Fase 1: Preparación (Se hace una sola vez)**

**1. En el ESP32 (Esclavo):**

* **Acción:** Carga (Flashea) el **main.ino** que te proporcioné (el que escucha por SerialPi en el UART 2).
* **Asegúrate:** De que todas tus librerías (motorControl.h, Control\_Recoleccion.h, etc.) estén en la misma carpeta o en la carpeta src de tu proyecto de Arduino/PlatformIO.

**2. En la Raspberry Pi (Maestro):**

* **Acción:** Copia todos tus scripts de Python (master\_controller.py, data\_logger.py, navigation\_node.py) a tu workspace de ROS 2.
  + **Ubicación:** ~/ros2\_ws/src/mi\_paquete/mi\_paquete/ (o como se llame tu paquete).
* **Acción:** Copia el archivo del modelo entrenado (navigation\_model.tflite) a esa misma carpeta.
* **Acción:** Instala el runtime de TensorFlow Lite en la Pi:

Bash

pip install tflite-runtime

* **Acción:** Compila tu workspace para que ROS "encuentre" tus nuevos scripts:

Bash

cd ~/ros2\_ws

colcon build

**3. Conexiones Físicas:**

* Conecta el **Lidar** a un puerto USB de la Pi.
* Conecta el **ESP32** a la Pi por **UART** (GND-GND, Pi TX -> ESP32 RX2, Pi RX -> ESP32 TX2).
* Conecta los **Drivers de Motor**, **Actuadores** (bombas, cepillos) y **Encoders** al ESP32 según tus archivos .h.

**Fase 2: Mapeo con SLAM (Opcional, pero recomendado)**

Si quieres que tu robot sepa dónde está (localización), necesitas un mapa.

* **Terminal 1 (Pi):** Lanza el driver del Lidar (asumiendo ROS 1 Noetic por tu OS de 32 bits).

Bash

roslaunch ydlidar\_ros D500.launch

* **Terminal 2 (Pi):** Lanza el SLAM (GMapping).

Bash

roslaunch mi\_aspiradora iniciar\_slam.launch

(Este es el archivo .launch que creamos antes).

* **Terminal 3 (Pi):** Lanza el control manual (teclado).

Bash

rosrun teleop\_twist\_keyboard teleop\_twist\_keyboard.py

* **Acción:** Conduce el robot por toda el área.
* **Finalizar:** Guarda el mapa (rosrun map\_server map\_saver -f mi\_mapa).

**Fase 3: Creación del Dataset (¡El paso crucial!)**

Aquí es donde le enseñas a la Red Neuronal a conducir.

* **Terminal 1 (Pi):** Lanza el driver del Lidar (ROS 2 Humble).

Bash

ros2 launch ldlidar\_stl\_ros2 ld19.launch.py

* **Terminal 2 (Pi):** Lanza el "puente" que mueve los motores (master\_controller.py).

Bash

source ~/ros2\_ws/install/setup.bash

ros2 run tu\_paquete master\_controller.py

* **Terminal 3 (Pi):** Lanza el control manual, **remapeando** el topic para que el logger lo escuche:

Bash

source ~/ros2\_ws/install/setup.bash

ros2 run teleop\_twist\_keyboard teleop\_twist\_keyboard --ros-args -r /cmd\_vel:=/cmd\_vel\_human

* **Terminal 4 (Pi):** Lanza el script de "grabación" (data\_logger.py).

Bash

source ~/ros2\_ws/install/setup.bash

ros2 run tu\_paquete data\_logger.py

* **Acción:** ¡Conduce! Muévete por el patio, esquiva obstáculos, sigue caminos. Conduce como te gustaría que el robot lo hiciera. Hazlo por 10-15 minutos.
* **Finalizar:** Ve a la **Terminal 4** (la del logger) y presiona Ctrl+C. Esto guardará el archivo dataset\_xxxx.npz.

**Fase 4: Entrenamiento (En tu PC)**

1. **Acción:** Copia el archivo dataset\_xxxx.npz de tu Pi a tu PC.
2. **Acción:** En tu PC, ejecuta el script de entrenamiento: python train\_model.py.
3. **Resultado:** Obtendrás el archivo navigation\_model.tflite.
4. **Acción:** Copia ese archivo navigation\_model.tflite de vuelta a la Pi (en la misma carpeta que tus scripts de ROS 2).

**Fase 5: ¡Operación Autónoma! (La Ejecución Final) 🚀**

Este es el momento de la verdad. Ya no usarás el teclado. La Red Neuronal conducirá.

**1. Enciende el Robot** (Pi y ESP32).

**2. Abre 3 Terminales en tu Pi:**

* **Terminal 1 (Lidar):**

Bash

source ~/ros2\_ws/install/setup.bash

ros2 launch ldlidar\_stl\_ros2 ld19.launch.py

* **Terminal 2 (Puente a Motores):**

Bash

source ~/ros2\_ws/install/setup.bash

ros2 run tu\_paquete master\_controller.py

* **Terminal 3 (El Cerebro NN):**

Bash

source ~/ros2\_ws/install/setup.bash

ros2 run tu\_paquete navigation\_node.py

**3. ¡Observa!**

* En la **Terminal 3**, verás los logs de la Red Neuronal publicando comandos /cmd\_vel.
* En la **Terminal 2**, verás los logs del puente enviando los comandos <M:...,...> al ESP32.
* **¡El robot debería empezar a moverse solo!**

**Para activar los actuadores (succión, cepillos, etc.):**

Abre una **Terminal 4** y envía los comandos manualmente:

* **Para encender la succión:**

Bash

ros2 topic pub /control/suction std\_msgs/msg/Bool "data: true" --once

* **Para encender los cepillos:**

Bash

ros2 topic pub /control/brushes std\_msgs/msg/Bool "data: true" --once

* **Para apagar todo:**

Bash

ros2 topic pub /control/suction std\_msgs/msg/Bool "data: false" --once

ros2 topic pub /control/brushes std\_msgs/msg/Bool "data: false" --once