

Instituto Tecnológico de Estudios Superiores de Monterrey



Implementación de robótica inteligente & Manchester Robotics

Manchester Robotics: Entrega Final

Profesores:

Rigoberto Cerino Jiménez

Dr. Mario Martinez

Integrantes

Daniel Castillo López A01737357

Emmanuel Lechuga Arreola A01736241

Paola Rojas Domínguez A01737136

13 de Junio de 2025

Índice

Índice.....	1
Resumen.....	2
Objetivos.....	3
Objetivo general.....	3
Objetivos específicos.....	3
Introducción.....	4
Unidades Funcionales.....	5
I. Introducción a las características del robot.....	5
Plataformas de Procesamiento.....	6
Sensores Integrados.....	6
Sistema de Actuación.....	6
Sistemas de Comunicación.....	7
La Hackerboard y su Interacción con el Puzzlebot.....	7
Arquitectura Técnica del Puzzlebot.....	8
II. Cinemática del Puzzlebot.....	8
Modelo cinemático diferencial.....	8
Consideraciones cinemáticas del Puzzlebot.....	9
III. Control Proporcional para un Robot Móvil Diferencial.....	10
IV. Navegación Autónoma del Puzzlebot.....	11
Seguimiento de línea.....	11
Máquina de estados para navegación.....	11
Lógica de giros y trayectorias especiales.....	12
Integración con visión y señales.....	12
V. Visión por computadora.....	12
Seguidor de línea.....	12
Detección de señales y semáforos con aprendizaje profundo.....	13
Integración visual en la arquitectura ROS.....	14
VI. Aprendizaje.....	14
Interacción con distintas fuentes de información.....	19
I. Arquitectura del sistema.....	20
II. Protocolo de comunicación externa.....	21
Comunicación mediante SSH.....	21
III. Protocolo de comunicación interna.....	22
Intercambio de información entre nodos.....	22
Comunicación entre Jetson y Hackerboard.....	23
Solución del problema.....	24
Resultados.....	28
Conclusiones.....	31
I. Conclusiones generales como equipo.....	31
II. Reflexión personal.....	32
Anexos.....	35
Referencias.....	36

Resumen

El resumen presenta una síntesis de los aspectos más relevantes del proceso de investigación y desarrollo llevado a cabo, destacando los objetivos, la metodología empleada y los principales resultados obtenidos a lo largo de la materia de Implementación de robótica inteligente junto con fundamentos de la robótica para la presentación final de este proyecto.

El seguimiento de líneas utilizando visión por computadora es una tarea común en la robótica móvil, y representa un paso importante hacia la autonomía total de un robot. En este reto, se trabajó con el Puzzlebot, un robot móvil diferencial, para que fuera capaz de detectar una línea en el suelo y seguirla, al mismo tiempo que interpreta señales y luces del semáforo a una escala óptima para el robot y gracias al entrenamiento de un modelo de redes neuronales reacciona al poder visualizar las mismas en tiempo real.

Para ello, fue necesario aplicar conocimientos adquiridos en retos anteriores, como la navegación punto a punto, la implementación de capas de decisión, la odometría, el control de motores, procesamiento de imagen entre otros temas relacionados. En esta ocasión se añadió una nueva capa: la percepción visual en conjunto con un modelo de red neuronal en yolo V8. El reto consistió en diseñar un sistema que combine estos elementos para obtener un comportamiento autónomo y fluido del robot, que pueda interpretar su entorno visual y actuar en consecuencia.

Además del seguimiento de línea, el robot debía responder a señales visuales específicas. El color rojo en un semáforo indica que debe detenerse; el amarillo, que debe disminuir su velocidad; y el verde, que puede continuar, ademas de las señales de transito como Stop, detenerse, give way y work para bajar velocidad, front (flecha hacia delante) para continuar en derecho, left (flecha hacia la izquierda) y right (flecha a la derecha) para girar a la izquierda y derecha. Todo esto debía ejecutarse sin intervención humana, asegurando que el sistema sea completamente autónomo y capaz de enfrentar condiciones reales como variaciones de luz o perturbaciones físicas.

El sistema debía ser robusto, por lo que se consideraron estrategias para mejorar su estabilidad, como el ajuste de parámetros del controlador, el filtrado de imágenes, preprocesado de imágenes y muestreo de las señales. Este reto permitió poner en práctica múltiples conceptos de robótica y visión artificial, integrando diferentes componentes en una solución funcional.

Objetivos

A continuación, presentamos el objetivo general y los objetivos particulares del reto, los cuales están enfocados en la programación de un Puzzlebot para comportarse como un vehículo autónomo.

Objetivo general

Desarrollar un sistema autónomo de navegación para el Puzzlebot que le permita recorrer una pista con forma de “b”, siguiendo una trayectoria definida y reconociendo señales de tránsito y semáforos utilizando visión por computadora, con el fin de simular un entorno de conducción real y demostrar habilidades adquiridas en percepción, control y autonomía móvil.

Objetivos específicos

- I. Implementar un sistema de detección visual para identificar señales de tránsito como “Giro a la derecha”, “Solo de frente”, “Ceda el paso”, “Alto”, entre otras, mediante algoritmos de visión o aprendizaje automático.
- II. Diseñar un módulo de detección de semáforos que permita identificar los estados de luz (rojo, amarillo, verde) y ajustar el comportamiento del robot en consecuencia.
- III. Mantener la trayectoria del robot centrada sobre la línea negra de la pista utilizando algoritmos de seguimiento de línea robustos.
- IV. Garantizar la navegación autónoma sin intervención humana, permitiendo que el robot recorra el circuito completo, incluyendo curvas e intersecciones, cumpliendo con las reglas impuestas por las señales y semáforos.
- V. Demostrar el funcionamiento del sistema mediante un video explicativo, donde se presentan los algoritmos utilizados, retos enfrentados, soluciones implementadas y diagramas del flujo de trabajo.

Introducción

En el contexto de la materia Implementación de Robótica Inteligente, se ha desarrollado un reto final orientado a integrar diversos conocimientos adquiridos a lo largo del curso, tales como percepción, control, odometría y navegación autónoma. Este reto plantea como objetivo principal el diseño de un sistema completo para un robot móvil tipo Puzzlebot, capaz de navegar de forma autónoma por una pista con forma de “b”, reconociendo señales de tránsito y luces de semáforo mediante visión por computadora.

El proyecto no solo involucra el seguimiento preciso de una trayectoria preestablecida, sino que también introduce una capa de percepción visual avanzada, haciendo uso de algoritmos de procesamiento de imagen y un modelo de red neuronal (YOLOv8) entrenado para la detección de señales específicas. Con ello, se busca simular un entorno de conducción real a escala, en el cual el robot sea capaz de interpretar su entorno visual y tomar decisiones adecuadas en tiempo real.

Además, se implementaron mecanismos para garantizar que el robot actúe sin intervención humana, ajustando su comportamiento según el reconocimiento de señales como “Alto”, “Ceda el paso”, “Giro a la izquierda/derecha” o luces de semáforo. Esto representa un ejercicio de integración práctica de varios componentes tecnológicos dentro del marco de un sistema embebido, reforzando así el aprendizaje activo, el trabajo en equipo y la capacidad de análisis en entornos reales y desafiantes.

Unidades Funcionales

Esta sección describe los componentes fundamentales que conforman el sistema robótico implementado en el reto de navegación autónoma. A lo largo del desarrollo del proyecto, se abordaron los subsistemas esenciales que permiten el funcionamiento coordinado del Puzzlebot, desde su estructura física y propiedades cinemáticas, hasta los algoritmos de control, navegación autónoma, percepción visual y aprendizaje automático. Cada uno de estos módulos desempeña un rol clave en la arquitectura general, habilitando al robot para desplazarse, de forma inteligente dentro de un entorno controlado, interpretar señales visuales y ejecutar maniobras dinámicas en función de las condiciones del entorno simulado. Esta integración de hardware y software refleja un enfoque a la construcción de sistemas robóticos autónomos, siguiendo principios de diseño contemporáneos en robótica móvil.

I. Introducción a las características del robot

El Puzzlebot es una plataforma de robótica móvil diseñada para el aprendizaje, la investigación y el desarrollo de algoritmos avanzados en navegación autónoma, visión por computadora e inteligencia artificial. Su arquitectura modular permite a los usuarios trabajar con diferentes configuraciones de hardware y software, adaptándose a distintos niveles de complejidad según la aplicación deseada. (Manchester Robotics Ltd, s. f.-f)

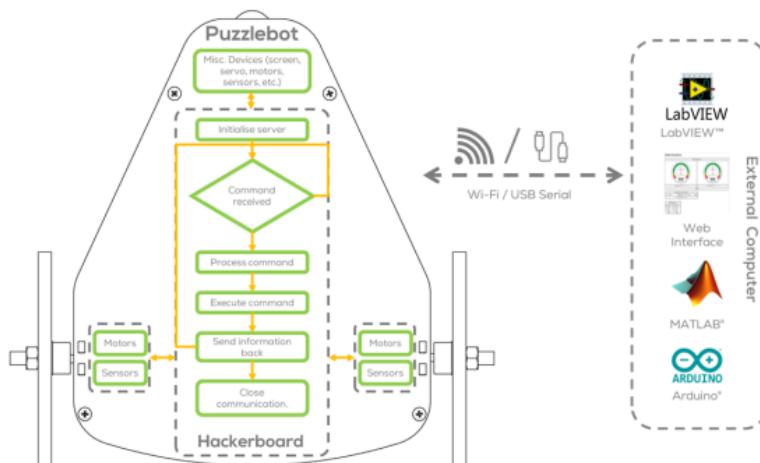


Figura 1: Diagrama de sistema de control

Como presenta la Figura 1 da una descripción detallada del robot móvil Puzzlebot, que está basado en una Hackerboard, que actúa como el procesador principal del sistema que está directamente conectado con los dispositivos periféricos como motores, sensores y otros dispositivos que se pueden agregar.

Plataformas de Procesamiento

El Puzzlebot cuenta con diferentes ediciones que emplean diversas plataformas de procesamiento, cada una con características específicas:

- **Hacker Edition:** Utiliza la Hackerboard, una plataforma especializada en procesamiento en tiempo real, ideal para el control de bajo nivel, la navegación, la evitación de obstáculos y la implementación de algoritmos de SLAM basados en LiDAR.
- **NVIDIA Jetson Edition:** Equipada con una Jetson Nano u Orin Nano, que en combinación con la Hackerboard, permite el desarrollo de algoritmos avanzados en inteligencia artificial y visión por computadora.
- **RPi Edition:** Basada en la Raspberry Pi 5, una plataforma versátil y asequible, idónea para el aprendizaje y el desarrollo de algoritmos de localización, planificación de rutas y navegación.

Sensores Integrados

El Puzzlebot está equipado con diversos sensores que le permiten recopilar información del entorno y mejorar su capacidad de navegación:

- **LiDAR:** Utilizado para la generación de mapas y la navegación autónoma.
- **Cámara Raspberry Pi:** Disponible en las ediciones Jetson y RPi, es utilizada para tareas de visión por computadora.
- **Sensor TOF (Time of Flight):** Presente en la edición Jetson, permite realizar mediciones de distancia con alta precisión.

Sistema de Actuación

El sistema de actuación del Puzzlebot está compuesto por motores conectados a la plataforma de procesamiento, lo que permite la aplicación de algoritmos de control para optimizar el movimiento del robot. En la Figura 2 se ilustra esta relación, destacando cómo los motores reciben instrucciones de la plataforma para ejecutar desplazamientos eficientes.

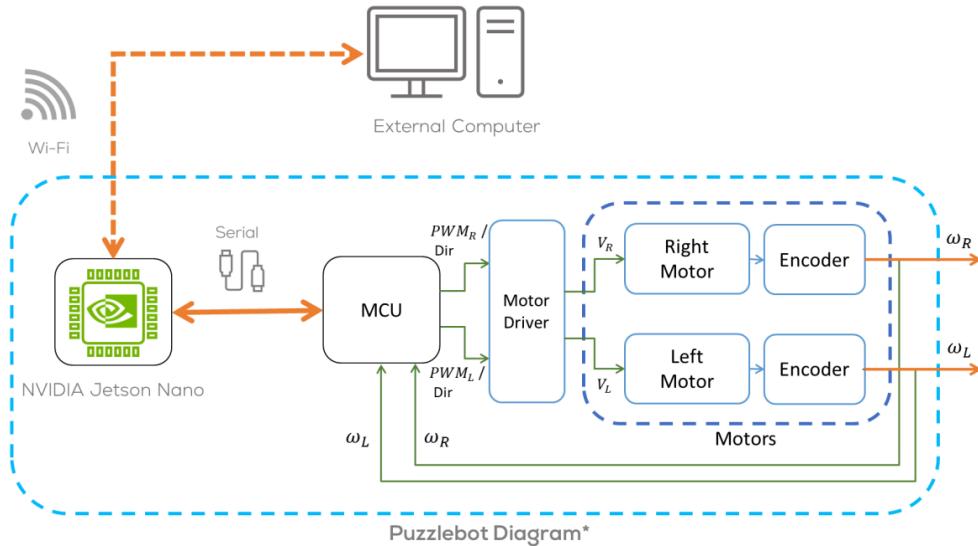


Figura 2. Diagrama de sistema del puzzlebot.

Sistemas de Comunicación

El Puzzlebot incorpora diferentes opciones para la comunicación y el control, facilitando su integración en diversos entornos de desarrollo:

- **Modo autónomo:** En esta configuración, el robot opera de manera independiente, ejecutando la programación preestablecida en su plataforma de procesamiento.
- **Wi-Fi y comunicación en serie:** Estas opciones permiten la interacción con computadoras externas y la integración con entornos de desarrollo como ROS, MATLAB y LabVIEW.

La Hackerboard y su Interacción con el Puzzlebot

La Hackerboard constituye el núcleo de control de bajo nivel del Puzzlebot, diseñada para procesamiento en tiempo real. Como se observa en la Figura 3, esta plataforma ejecuta funciones clave como el control de motores, la evitación de obstáculos y la implementación de SLAM en dos dimensiones utilizando LiDAR.

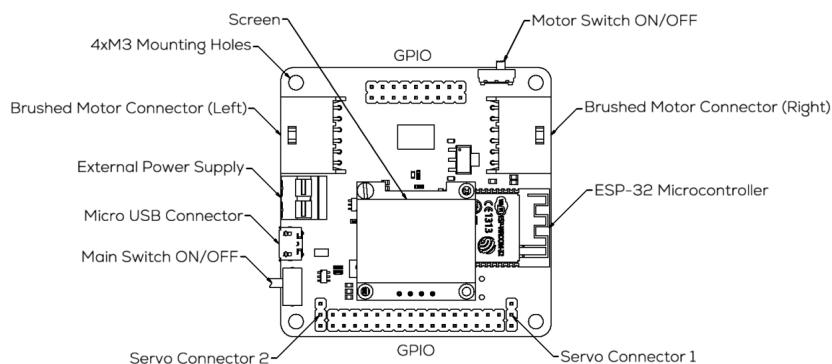


Figura 3. Hackerboard.

La Hackerboard permite dos modos de operación principales:

1. **Modo autónomo (Standalone):** En esta configuración, la Hackerboard ejecuta el código cargado por el usuario mediante el entorno de desarrollo Arduino IDE, utilizando bibliotecas de MCR2 para la interacción con sensores y actuadores.
2. **Modo externo:** En este modo, la Hackerboard se comunica con una computadora externa o con una plataforma de alto rendimiento (Jetson o RPi) a través de USB Serial o Wi-Fi. Transmite datos de los sensores, como las lecturas de los encoders, y recibe comandos de movimiento que se aplican a los motores.

Arquitectura Técnica del Puzzlebot

La arquitectura del Puzzlebot está compuesta por diversas entradas y salidas que permiten la interacción con su entorno y su integración en sistemas de control más complejos en la Figura 4 se pueden apreciar algunos de estos.

- **Entradas/Sensores:** Incluye encoders, LiDAR, cámaras y sensores TOF.
- **Salidas/Actuadores:** Comprende motores DC, servomotores y LEDs.
- **Protocolos de comunicación:** Emplea UART y SPI para la comunicación con periféricos, así como ROS Serial para la integración con el sistema ROS.

II. Cinemática del Puzzlebot

El Puzzlebot es un robot móvil diferencial, lo que significa que su locomoción se basa en el movimiento independiente de dos ruedas motrices montadas sobre un mismo eje. Cada rueda está conectada a su propio motor, y en la parte posterior se ubica una tercera rueda libre cuyo objetivo es proporcionar estabilidad sin intervenir en el movimiento.

Modelo cinemático diferencial

Para modelar el comportamiento del Puzzlebot en el plano, se utiliza un modelo cinemático diferencial directo. Este modelo permite relacionar las velocidades angulares de las ruedas con las velocidades lineales y angulares del robot sin considerar dinámicas como fuerzas, fricción o momentos de inercia, asumiendo que el robot es una masa puntual ubicada en el centro del eje que une a las ruedas motrices.

La velocidad lineal y la velocidad angular del robot se obtienen como funciones de las velocidades angulares de las ruedas y parámetros físicos del robot:

$$V = \frac{r(\omega_R + \omega_L)}{2} \quad (1)$$

$$\omega = \frac{r(\omega_R + \omega_L)}{l} \quad (2)$$

Donde:

- V : velocidad lineal del robot

- ω : velocidad angular del robot
- r : radio de las ruedas (en metros)
- l : distancia entre las ruedas (en metros)
- ω_R : velocidad angular de la rueda derecha
- ω_L : velocidad angular de la rueda izquierda. (Castillo et al., s. f.)

A partir de estas velocidades se puede estimar la velocidad del punto de control en el sistema de coordenadas global (x , y , θ) mediante:

$$x' = V \cdot \cos(\theta) \quad (4)$$

$$y' = V \cdot \sin(\theta) \quad (5)$$

$$\theta' = \omega \quad (6)$$

Estas ecuaciones representan la cinemática directa del robot y permiten conocer su movimiento a partir de la información de las ruedas. Para estimar la pose (posición y orientación) del robot en un instante dado, se requiere integrar estas ecuaciones numéricamente a lo largo del tiempo utilizando un método como Euler:

$$x_{k+1} = x_k + V_k \cdot \cos(\theta_k) \cdot \Delta t \quad (7)$$

$$y_{k+1} = y_k + V_k \cdot \sin(\theta_k) \cdot \Delta t \quad (8)$$

$$\theta_{k+1} = \theta_k + \omega_k \cdot \Delta t \quad (9)$$

Donde Δt es el tiempo de muestreo entre cada medición. Este enfoque, conocido como odometría, es utilizado para estimar la trayectoria del robot a partir de los encoders integrados en el Puzzlebot.

Consideraciones cinemáticas del Puzzlebot

Los valores típicos para los parámetros físicos del Puzzlebot de Manchester Robotics en su configuración estándar son:

- $r = 0.05$ m (radio de las ruedas)
- $l = 0.19$ m (distancia entre ruedas)
- $\Delta t \approx 0.1$ s (tiempo de muestreo asumido).

Estas características permiten que el robot tenga una maniobrabilidad suficiente para realizar tareas de navegación en entornos estructurados, como el seguimiento de líneas o el desplazamiento entre puntos específicos. El modelo cinemático es también la base para el diseño de controladores de velocidad y trayectoria.

El modelo diferencial tiene como ventaja su simplicidad y aplicabilidad a entornos reales con bajo costo computacional, siendo ideal para sistemas embebidos como la Hackerboard del

Puzzlebot. Sin embargo, también presenta limitaciones, como la acumulación de error en odometría y la imposibilidad de alcanzar ciertos puntos directamente debido a sus restricciones no holonómicas.

La compresión del sistema cinemático es fundamental para el desarrollo de algoritmos de navegación autónoma, ya que permiten comprender cómo traducir los comandos de velocidad, los enviados al tópico `/cmd_vel` en ROS, en desplazamientos físicos del robot, con la precisión necesaria para interactuar con señales de tránsito y semáforos mientras se sigue una trayectoria.

III. Control Proporcional para un Robot Móvil Diferencial

El controlador proporcional (P) es especialmente útil en sistemas donde el objetivo es minimizar un error de seguimiento en función de una referencia deseada. En el contexto del Puzzlebot, un robot diferencial, el controlador proporcional se implementó para regular la dirección del robot mediante corrección continua de su orientación con base en el error visual estimado a partir de una línea guía captada por visión por computadora.

Como se mencionó anteriormente, el robot diferencial presenta una arquitectura de dos ruedas motrices dispuestas en un mismo eje, permitiendo controlar su trayectoria ajustando las velocidades relativas de ambas ruedas. Esta estructura es particularmente adecuada para implementar controladores de tipo cinemático, como el control P, ya que permite una respuesta rápida y precisa a cambios en el error de orientación.

En este sistema, el error de entrada al controlador se define como la desviación horizontal del centroide de la línea negra detectada respecto al centro de la imagen. Este error visual denotado como e , es directamente proporcional al grado de desviación lateral del robot respecto a su trayectoria deseada. La salida del controlador es una velocidad angular ω , calculada mediante:

$$\omega = -K_p \cdot e \quad (10)$$

donde K_p es la constante de ganancia proporcional ajustable. Este valor se determina empíricamente y permite controlar la sensibilidad del robot ante el error: un K_p mayor genera una respuesta más rápida pero también puede inducir oscilaciones o comportamiento inestable, mientras que un valor demasiado bajo resulta en un seguimiento lento e impreciso. Este esquema de control se complementa con una velocidad lineal fija V .

Además, el controlador fue probado y ajustado en ejecución real sobre una pista física, comprobando su capacidad para mantener la trayectoria centrada sobre la línea negra, incluyendo trayectorias rectas, curvas y secciones con señales que modifican el comportamiento del robot.

El uso del control P en este contexto demuestra que, a pesar de su simplicidad, puede ofrecer resultados robustos cuando se combina con una adecuada arquitectura modular de percepción

y control. La implementación en ROS 2 facilita su integración con otros nodos de procesamiento, asegurando una navegación fluida y autónoma.

IV. Navegación Autónoma del Puzzlebot

La navegación autónoma del Puzzlebot en este reto se implementó como un sistema de percepción-reactivo, basado principalmente en visión por computadora y control cinemático diferencial. El objetivo fue recorrer de forma autónoma una pista con trayectoria en forma de “b”, siguiendo una línea negra en el suelo como guía principal, integrando la respuesta ante señales de tránsito en intersecciones reguladas por semáforos.

Seguimiento de línea

El sistema de navegación utiliza una técnica de seguimiento de línea basada en visión que permite calcular un error de posición transversal respecto a una trayectoria deseada. Este error es calculado a partir del procesamiento de la imagen capturada por una cámara colocada en la parte frontal del robot, la cual segmenta la región inferior de la imagen para detectar contornos y calcular centroides de la línea en tres regiones verticales.

El segundo centroide es utilizado como referencia principal para el control. Su posición horizontal respecto al centro de la imagen define el error normalizado, el cual se transforma directamente en una velocidad angular mediante un controlador proporcional, como se explicó en la sección anterior. Esto permite mantener el robot alineado con la línea de manera continua.

El sistema también incluye una lógica de recuperación automática (modo recover) en caso de pérdida del centroide principal, utilizando la información histórica de los centroides previos o, si es necesario, del centroide más bajo disponible. Esto permite al robot mantener su orientación general incluso ante interrupciones breves de visibilidad de la línea, como aquella causada por las intersecciones en la pista.

Máquina de estados para navegación

La navegación se basa además en máquinas de estados implementadas en el nodo controller la cual decide el comportamiento del robot en función de múltiples fuentes de entrada:

- El error de seguimiento de línea
- El estado del semáforo
- Las señales de tránsito detectadas

La máquina de estados del semáforo permite cambiar entre modos de navegación, tales como:

- STOP: inmovilidad ante semáforo rojo
- SLOW: reducción de velocidad ante semáforo amarillo
- GO: seguimiento de línea a velocidad normal

Por otro lado cada señal activa una rutina de control diferente:

- stop: el robot se detiene completamente hasta dejar de ver la señal
- ahead: avance temporal recto ante la señal de frente
- turn: ejecución secuencial de movimiento ante giros detectados avance inicial, giro de 90° hacia el lado indicado por la señal y avance final
- roadwork/give way: reducir la velocidad al 50%

Estos comportamientos están definidos de forma estructurada, permitiendo una transición fluida entre estados y manteniendo la coherencia con las reglas de tráfico simuladas.

Lógica de giros y trayectorias especiales

Uno de los elementos clave de la navegación en este reto fue la ejecución de trayectorias compuestas en respuesta a señales de giro. Estas se implementaron mediante una secuencia temporal de acciones:

1. Avance inicial de 30-40 cm
2. Giro de 80-90 grados, ajustado dinámicamente según el error visual
3. Avance final de 20 cm

Asimismo este comportamiento se ajusta automáticamente en función del estado del semáforo: si se detecta luz amarilla, el robot avanza más lento; si es roja el comportamiento se pausa; si es verde continua normalmente. Esta lógica es robusta frente a variaciones ambientales y permite una navegación autónoma sensible al contexto del entorno.

Integración con visión y señales

El sistema de navegación está completamente integrado con el sistema de percepción visual basado en YOLOv8. Las señales como “turn left”, “turn right”, “stop”, “ahead”, “give way” y “roadwork” son detectadas visualmente y enviadas como códigos discretos al controlador. Este interpreta la señal y ejecuta la acción correspondiente, modificando la navegación en tiempo real. La sincronización entre la detección visual y la navegación es crucial para mantener un comportamiento autónomo coherente y seguro.

V. Visión por computadora

La percepción visual fue el eje central del sistema autónomo desarrollado para el Puzzlebot. En este reto, se implementaron múltiples algoritmos de visión por computadora que permiten al robot detectar tanto la línea guía de navegación como diversas señales de tránsito y estados del semáforo, utilizando una arquitectura distribuida en ROS 2. La solución combina técnicas tradicionales de procesamiento de imagen con aprendizaje profundo, lo cual permite adaptar el comportamiento del robot a su entorno de manera dinámica y precisa.

Seguidor de línea

El sistema de seguimiento de línea implementado utiliza una cámara RGB montada en el Puzzlebot, orientada hacia el piso. La imagen capturada es procesada por el nodo

camara_node.py, que recorta una región de interés (ROI) en la parte inferior de la imagen y aplica un umbral binario sobre la escala de grises para aislar la línea negra del fondo. Posteriormente, mediante operaciones morfológicas de apertura, se eliminan artefactos pequeños y ruido.

El nodo line_follower.py analiza la imagen procesada utilizando detección de contornos. El contorno más grande se asume como la línea principal, y se calculan tres centroides correspondientes a tres divisiones horizontales del ROI. Estos centroides se utilizan para calcular el error lateral del robot respecto al centro de la imagen, el cual es normalizado y enviado al tópico /error como entrada para el controlador proporcional. La lógica incluye un modo de recuperación (recover) en el que, si se pierde uno o más centroides debido a interrupciones en la línea, el sistema utiliza información pasada o niveles inferiores de la imagen para mantener una estimación aproximada del error y permitir la reconexión con la trayectoria. Esta implementación demostró ser efectiva incluso ante condiciones variables de iluminación.

Detección de señales y semáforos con aprendizaje profundo

La detección de señales visuales y semáforos fue implementada utilizando un modelo de redes neuronales convolucionales YOLOv8, entrenado específicamente con una base de datos propia que incluye clases como:

- Semáforo: rojo, verde, amarillo (S_rojo, S_verde, S_amarillo),
- Señales de tránsito: right, left, front, stop, giveaway, work.

El modelo fue ejecutado en el nodo detection.py, el cual suscribe imágenes comprimidas desde /compressed_image, realiza inferencia en tiempo real, y publica los resultados codificados en los tópicos /color (para semáforo) y /sign (para señales).

Para mejorar la robustez del sistema, se incorporó una lógica de detección persistente, que exige que una clase se mantenga visible durante al menos 0.5 segundos antes de ser confirmada como válida. También se filtraron las detecciones por área mínima para evitar falsos positivos de objetos pequeños o parcialmente visibles.

El sistema demostró ser capaz de detectar múltiples señales en una sola imagen, registrar su clase, posición y confianza, y emitir comandos coherentes al sistema de control. La combinación de este modelo con una frecuencia de muestreo de ~30 Hz permitió que el Puzzlebot reaccionara de forma oportuna a cambios en el entorno y modificara su comportamiento (como detenerse, girar o avanzar lento) de acuerdo con las reglas del entorno simulado.

Integración visual en la arquitectura ROS

Ambos subsistemas de visión se integran dentro del entorno ROS 2 mediante tópicos claramente definidos. Esta separación lógica permite la reutilización de nodos, la mejora independiente de módulos y la trazabilidad del sistema. Además, se implementó una salida visual en tiempo real que permite grabar la evolución del entorno mediante debug_image y videos de detección con anotaciones, lo cual facilitó la validación del sistema y el análisis de fallas.

VI. Aprendizaje

Recopilación de datos para entrenamiento.

La recopilación de datos se realizó implementando un código con el cual podíamos ver a través de la cámara del puzzlebot y al momento de presionar la letra S, capturamos la imagen y se guardaba de manera automática en nuestra carpeta del proyecto, el número límite de este código fueron 150 imágenes, de las cuales íbamos cambiando las señales de posición con el fin de recabar los datos suficientes para el modelo, este proceso se repitió varias veces debido al entrenamiento del modelo.

Segmentación del dataset.

Los datos recopilados en este caso imágenes se segmentan mediante el uso de Robo Flow, donde se puede hacer como un bounding box en el cual podemos etiquetar el objeto que queremos que le de prioridad de identificación en nuestro caso fue en base a las señales viales y al semáforo. En lo cual se tomó el total de las imágenes y por medio de un código implementado en python pudimos aumentar, debido a que el código procesaba las imágenes agregando ruido gaussiano de un valor de 10, 20 y 30, con el fin de aumentar la precisión de detección del modelo ante las señales de tránsito y semáforo; una vez obtenidas las imágenes con ruido gaussiano procedemos a etiquetarlas y a añadirlas a la base de datos completa.

Entrenamiento del modelo con redes neuronales.

Esta solución como se meconio antes, parte de la toma de diversas fotos de los objetos utilizados como el semáforo y señales, y creación del dataset en la página de roboflow que cataloga en diferentes clases las fotografías tomadas, y genera una dataset mayor con cambios en la imagen para mejorar su entrenamiento, nuestra versión contiene 9959 de imágenes en total. Ya con la dataset podemos pasar a entrenar el modelo.

- Primero instalamos las librerías necesarias para “YOLOv 8” y “roboflow” para la gestión del dataset,
- Importamos las librerías de los módulos de python como lo son “torch” y IPython.display para utilizar la GPU y tener un entrenamiento más eficiente y eficaz.
- Utilizamos la librería de robo flow para conectarse a nuestra cuenta y descargar un dataset específico (mcr-final-challenge versión 2) en el formato de YOLO v8.

- Mostramos el contenido del archivo .yaml del dataset con el fin de observar el número de clases y los nombres, para confirmación de las mismas.
- Creamos una función auxiliar llamada “writetemplate” para facilitar la escritura de archivos utilizando plantillas y variables de python.
- Iniciamos un modelo tipo YOLOv8, pre-entrenado, para obtener la versión small que es yolov8s .pt
- Empezamos el entrenamiento con el método model.train() donde se necesitan los siguientes parámetros como lo es Data que es la ruta del archivo .yaml del dataset.
- Ocupamos algunos datos como Epochs que es el número de épocas de entrenamiento en nuestro caso son 70, batch que es el tamaño del batch que es de 16 y imgsz que es el tamaño de las imágenes para el entrenamiento, el nombre de la carpeta donde se guardarán los resultados.
- Utilizamos la función files.download de google colab para descargar los archivos de pesos del modelo entrenado, tanto el que obtuvo mejor rendimiento (best.pt) como el de la última época (last.pt).
- Mostramos la matriz de confusión generada durante el entrenamiento para evaluar visualmente el rendimiento del modelo por clase.
- Mostramos una imagen de un lote de validación con las predicciones del modelo para una inspección visual.
- Para finalizar ejecutamos un comando para validar el modelo entrenado en el conjunto de validación, lo que genera métricas de rendimiento más detalladas.

Resultados del modelo de red neuronal (validación).

- Primero descargamos la librería de “Ultralytics” e importamos Yolo; además de importar algunas funciones de imagen para mostrar en pantalla los resultados, como es loss, métricas, predicciones, pr curve y matriz de precisión.

Gráfico de pérdida y métricas.

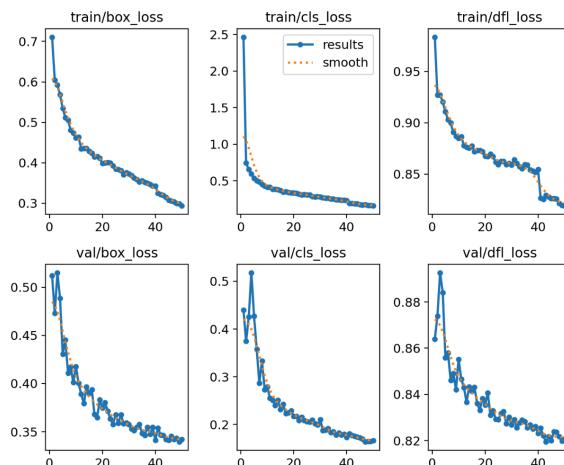


Figura 4. Gráfico de pérdidas (error).

- *box loss*: Qué tan bien predice las cajas (bounding boxes).
- *cls loss*: Qué tan bien clasifica los objetos detectados.
- *obj loss*: Qué tan bien detecta la presencia o ausencia de un objeto.

Como podemos observar en la Figura 4, estas curvas bajan conforme pasan las épocas esto nos dice que entre menor sea la valuación del error es un mejor aprendizaje (menos error = mejor aprendizaje).

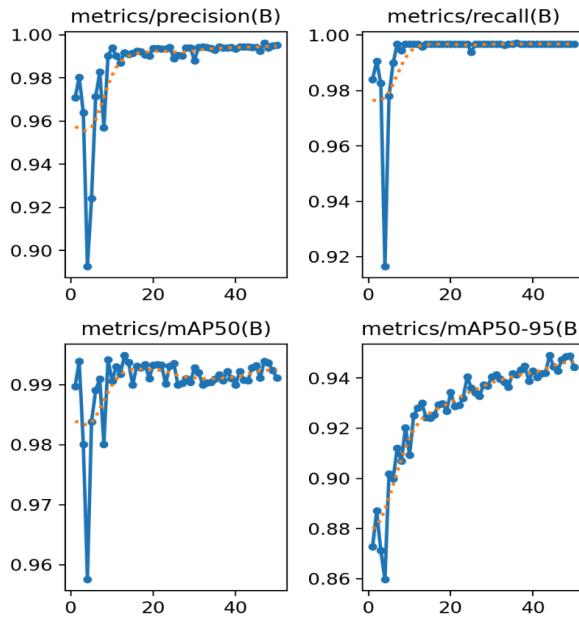


Figura 5. Gráfico de métricas de precisión y Recall.

Como podemos observar en las métricas de la Figura 5, estas tienen una gran precisión y una gran exactitud con pruebas reales, en cuanto a estas entre mayor sea el valor de la métrica podemos ver que tiene más exactitud de reconocer correctamente una señal y objeto.

- *Precisión*: De las detecciones que el modelo hizo.
- *Recall*: De los objetos reales detectados por el modelo.
- *mAP50* y *mAP50-95*: Medidas globales de rendimiento que combinan precisión y recall.

Curvas de precisión-Recall

- Precisión (Precisión): Porcentaje de las detecciones que fueron correctas.
- Recall (Exhaustividad): Porcentaje de los objetos reales logré detectar.
- El modelo refleja un buen equilibrio entre detectar todo lo posible (recall alto) y evitar falsos positivos (precisión alta).
- El área bajo la curva (AUC) indica el rendimiento general: cuanto más cerca esté la curva de la esquina superior derecha, mejor.
- Si una curva está muy hacia abajo o hacia la izquierda, el modelo está fallando para esa clase tenemos dos opciones una es que no la detecta (bajo recall), o que comete

muchos errores (baja precisión), como se puede ver en la Figura 6, el modelo cumple con la parte de estar muy bien entrenado y demostrarnos como se debe de ver una curva con un alto índice de validación.

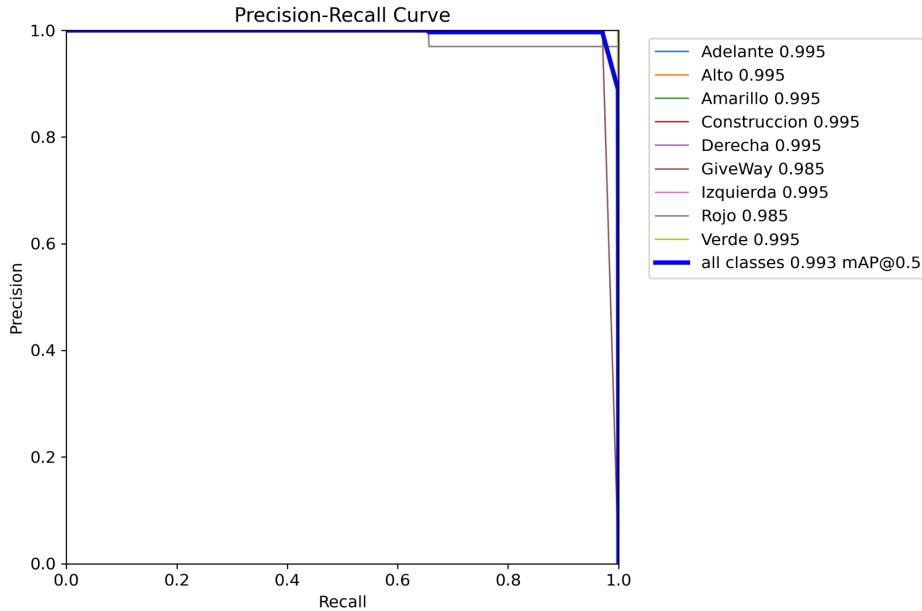


Figura 6. Curva de precisión/Recall

Esta se realiza con el fin de saber si el modelo es mejor detectando unas clases que otras, al igual si hay baja precisión en dado caso debemos de dar más ejemplos.

Matriz de confusión

- Es una tabla que compara las predicciones del modelo con las etiquetas reales para cada clase, su objetivo es mostrar.
 - Qué tan bien está clasificando cada clase.
 - Qué errores comete y con qué frecuencia.
- Diagonal (en color azul): son las predicciones correctas. El modelo acertó.
- Fuera de la diagonal: son errores donde el modelo confundió clases.

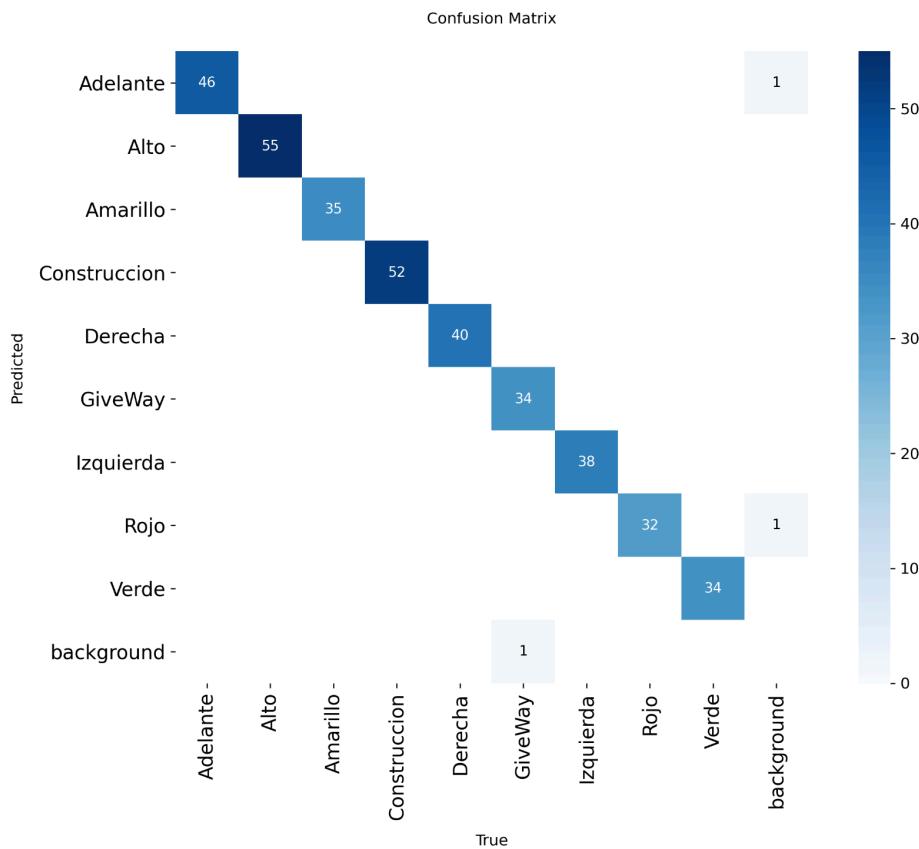


Figura 7. Matriz de confusión.

Como podemos ver en la Figura 7, el modelo evalúa correctamente en su mayoría todas las señales, excepto una vez en algunas de ellas, sin embargo como se puede observar es un margen demasiado poco como para que afecte a nuestro modelo.

Predicciones del batch de validación.

Con las predicciones del batch de validación podemos observar cómo el modelo se comporta con imágenes reales del conjunto de validación, es decir, datos que no se usaron durante el entrenamiento, pero que sí están etiquetados.

El modelo toma un *batch* (un conjunto de imágenes del conjunto de validación), realiza predicciones y muestra lo siguiente como se podrá ver a continuación en la Figura 8:

1. Cuáles objetos detectó.
2. A qué clase los asignó.
3. Con qué confianza (score).
4. Dónde están localizados (bounding boxes).

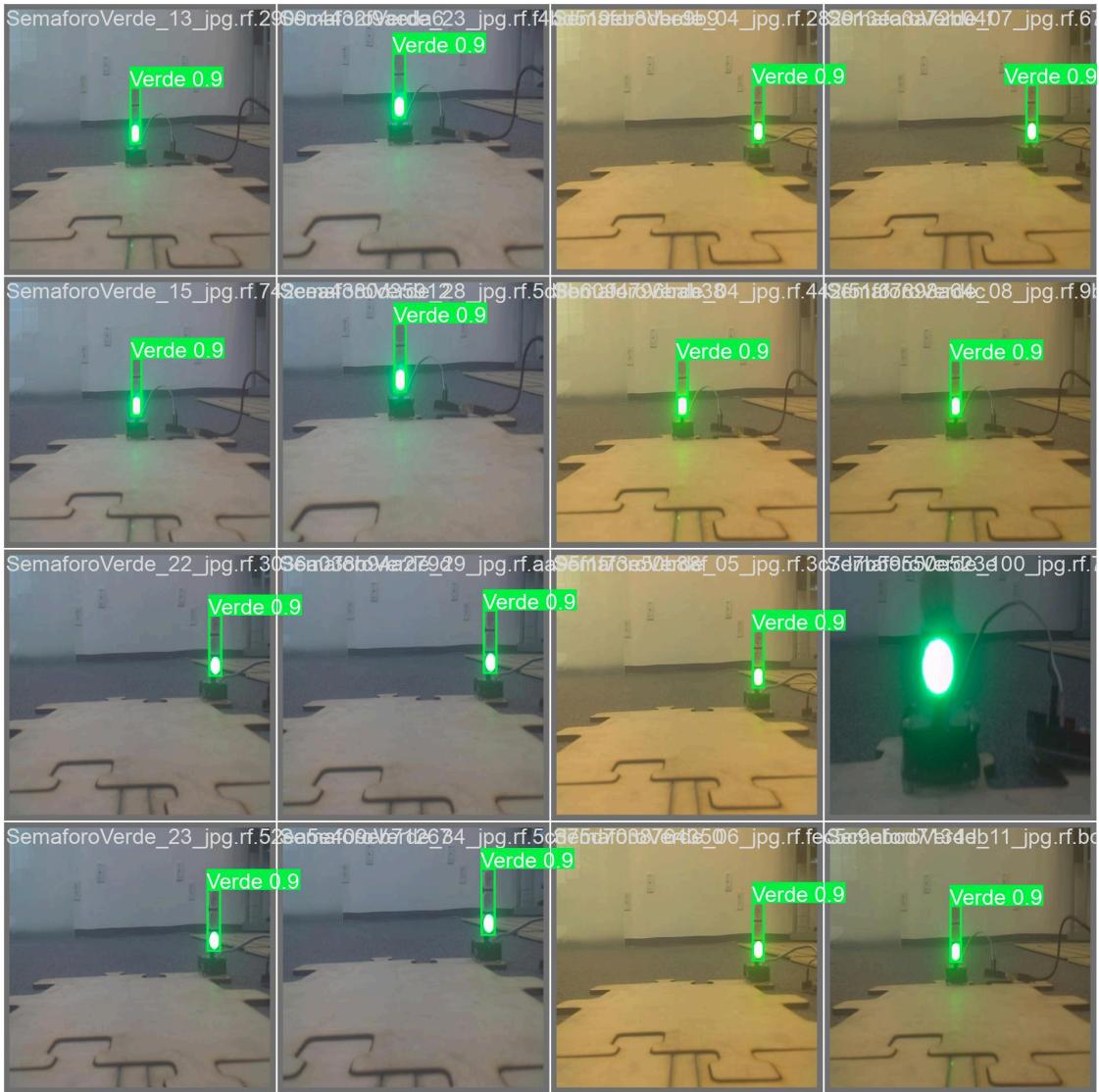


Figura 8. Batch de validación.

En nuestro caso podemos observar en la Figura 8 cómo detecta en su mayoría de los casos lo que es el semáforo, este es un ejemplo pero en sí detecta muy bien las demás señales.

Interacción con distintas fuentes de información

La interacción con fuentes de información es un componente fundamental en el diseño de sistemas robóticos autónomos, ya que permite integrar distintos módulos funcionales a través de una arquitectura distribuida. En el caso del Puzzlebot, dicha interacción se logra mediante el uso del middleware ROS 2, que habilita una comunicación estructurada entre nodos mediante tópicos, servicios y acciones. Esta arquitectura modular permite la coordinación entre sensores, actuadores y algoritmos de decisión, facilitando la ejecución de tareas complejas como la navegación autónoma, el seguimiento visual de trayectorias y la reacción a señales del entorno. En esta sección se describen los aspectos técnicos relacionados con la

arquitectura del sistema, los protocolos de comunicación utilizados y su implementación tanto a nivel interno (entre nodos) como externo (entre plataformas de procesamiento).

I. Arquitectura del sistema

El sistema autónomo desarrollado para el Puzzlebot está organizado como una arquitectura distribuida modular basada en el ROS 2. Esta plataforma permite la creación de nodos independientes que se comunican mediante una interfaz de paso de mensajes asincrónico, facilitando la integración de componentes como controladores, sensores, algoritmos de visión y módulos de aprendizaje automático.

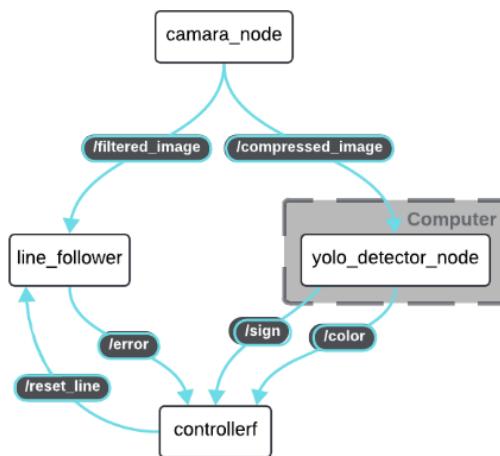


Figura 9: Interconexión del sistema

Tal y como se muestra en la Figura 9, la arquitectura está compuesta por cuatro nodos principales:

- **camara_node:** Nodo encargado del preprocesamiento de imagen y generación de la imagen binaria de delimita la línea negra. También publica una versión comprimida de la imagen RGB para el uso de sistema de detección.
- **line_follower:** Nodo de percepción que detecta la línea negra en la pista, estima el error transversal del robot y lo publica al sistema de control.
- **yolo_detector_node:** Nodo de visión por computadora que ejecuta un modelo Yolov8 para la detección de señales de tránsito y semáforos. Publica códigos discretos en tópicos específicos para su interpretación por el sistema de control.
- **controllerf:** Nodo de control y comportamiento. Integra la información del error visual, señales visuales y estado del semáforo. Implementa un controlador proporcional, una máquina de estados y lógica secuencial para giros, paradas y recuperaciones.

El flujo de información en el sistema se organiza en capas:

- Capa de percepción. La cámara envía imágenes al nodo de filtrado (**camara_node**), que produce dos salidas: una imagen binaria para el seguimiento de línea y una

imagen comprimida para detección. Estas imágenes fluyen hacia los nodos `line_follower` y `yolo_detector_node`, respectivamente.

- Capa de interpretación visual. El nodo `line_follower` publica errores visuales en el tópico `/error` mientras que el nodo de detección emite códigos en `/color` (semáforo) y `/sign` (señales de tránsito).
- Capa de decisión y control. El nodo `controllerf` recibe los mensajes publicados por los otros nodos y genera comandos de velocidad (`/cmd_vel`) así como señales auxiliares como `/reset_line`.

Gracias a esta arquitectura basada en ROS 2, el sistema es capaz de tolerar pérdidas parciales de información mediante mecanismos internos de persistencia o recuperación. Además el diseño modular permite extender la arquitectura fácilmente con nuevas fuentes de información, como sensores adicionales o algoritmos de planeación.

II. Protocolo de comunicación externa

Durante la implementación del sistema autónomo del Puzzlebot, se estableció una arquitectura distribuida entre la plataforma física del robot y una computadora externa. Esta separación permitió descargar parte de la carga computacional, principalmente la ejecución del modelo de detección basado en YOLOv8, en un equipo con mayores recursos, manteniendo la funcionalidad del robot sin comprometer el rendimiento en tiempo real.

Aunque la Jetson Nano está equipada con capacidad de procesamiento para tareas de visión, el modelo YOLOv8 fue ejecutado en una computadora externa para disminuir la carga computacional de la Jetson. Esta computadora actuó como parte del sistema ROS 2, conectándose a la misma red local y participando como un nodo adicional dentro del graph de ROS.

El nodo de detección fue configurado para suscribirse al tópico `/compressed_image`, publicado desde el robot, procesar las imágenes con el modelo de detección, y reenviar las señales detectadas a través del sistema ROS compartido. Esta estrategia descentralizada de ejecución permite aprovechar los recursos de hardware de forma eficiente, minimizando latencias perceptuales mientras se mantienen las decisiones de control centralizadas en el robot.

Comunicación mediante SSH

Para la transferencia de archivos, despliegue de código y control remoto de la plataforma Jetson Nano integrada en el Puzzlebot, se utilizó el protocolo SSH (Secure Shell). Esta comunicación cifrada permitió:

- Subir modelos entrenados (.pt) y scripts Python desde una computadora de desarrollo al entorno Linux de la Jetson.
- Ejecutar remotamente los nodos ROS 2 en la Jetson desde una terminal externa.
- Realizar pruebas, debugging y sincronización de código en tiempo real.

La comunicación SSH se estableció mediante una red local inalámbrica, lo que permitió trabajar de forma flexible sin requerir conexión física directa al robot. Esto fue especialmente útil en etapas de prueba y grabación de vídeo del sistema en funcionamiento.

Para mantener la interoperabilidad entre dispositivos, ambos sistemas (la Jetson en el robot y la computadora externa) compartieron el mismo espacio de red y entorno ROS 2 bajo el mismo dominio de configuración (ROS_DOMAIN_ID). La sincronización temporal y la compatibilidad entre nodos se garantizaron utilizando versiones compatibles de Python, rclpy, y dependencias de visión (cv2, ultralytics, cv_bridge), asegurando una operación coherente y sincrónica. (Manchester Robotics Ltd, s. f.-f)

III. Protocolo de comunicación interna

El sistema de navegación autónoma del Puzzlebot se estructura internamente sobre la arquitectura de ROS 2 (Robot Operating System 2), utilizando un modelo de comunicación publicador–suscriptor para intercambiar información entre nodos funcionales. Esta arquitectura permite desacoplar los distintos módulos de percepción, control y actuación, facilitando tanto la escalabilidad del sistema como su mantenimiento.

Intercambio de información entre nodos

Los diferentes módulos del sistema están organizados como nodos ROS 2 independientes que interactúan a través de tópicos específicos. Cada tópico transmite un tipo de mensaje estandarizado según el paquete std_msgs, sensor_msgs o geometry_msgs. Los mensajes clave intercambiados incluyen:

- /filtered_image (sensor_msgs/Image): imagen binaria procesada desde la cámara, publicada por camara_node.
- /compressed_image (sensor_msgs/CompressedImage): imagen RGB comprimida, utilizada por el modelo YOLO en yolo_detection_node
- /error (std_msgs/Float32): error transversal del robot respecto a la línea, generado por line_follower.
- /color (std_msgs/Int32): estado del semáforo detectado, codificado como 1 (rojo), 2 (verde), 3 (amarillo).
- /sign (std_msgs/Int32): código correspondiente a la señal visual detectada (turn, stop, ahead, etc.).
- /cmd_vel (geometry_msgs/Twist): comando de velocidad lineal y angular enviado al robot desde controllerf.
- /reset_line (std_msgs/Bool): señal enviada desde el controlador al seguidor de línea para reiniciar el historial de centroides.

Este sistema asincrónico permite que cada nodo se ejecute de manera independiente, reciba únicamente la información que necesita y emita su salida sin interferir directamente con los demás, mejorando la robustez general del sistema.

Comunicación entre Jetson y Hackerboard

La interacción entre la plataforma de procesamiento (Jetson Nano) y la Hackerboard del Puzzlebot se realizó utilizando una conexión USB serial, gestionada por el paquete rosserial. Esta interfaz permite enviar los comandos de velocidad generados por el nodo controllerf.py directamente al microcontrolador encargado del control de bajo nivel de los motores.

Este protocolo de comunicación interna garantiza que las decisiones tomadas a alto nivel en ROS 2 puedan traducirse en acciones físicas ejecutadas por el hardware del robot, cerrando así el lazo de control.

ROS 2 proporciona soporte para comunicación segura y robusta. En esta implementación, la sincronización entre nodos se mantuvo de forma natural al trabajar en una red local cerrada, con baja latencia y sin necesidad de sincronización de reloj externa. El sistema pudo operar con una frecuencia de muestreo estable (~30 Hz) en todos los nodos críticos, incluyendo percepción, control y visión.

Solución del problema

En esta parte desarrollaremos una solución para nuestro reto, en el cual consiste en poder tener una distinción en la visión de computadora, para poder tener el seguidor de línea y el análisis de colores del semáforo y esto implementado en conjunto con el modelo neuronal con el fin de tener una consecuencia directa al controlador del puzzlebot en base a como reacciona con las señales. Con el objetivo ya planteado, podemos realizar la siguiente conexión de Nodos en la Figura 10 para tener orden en la ejecución de nuestro problema.

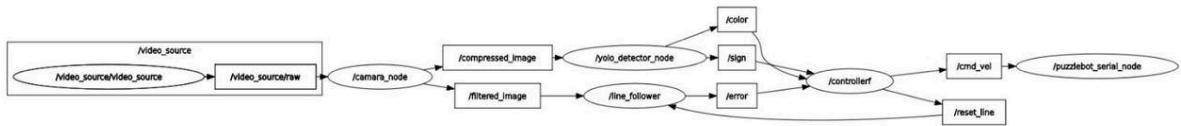


Figura 10: Diagrama de conexión de nodos

Nodo de filtrado de cámara (filter_node):

Ahora, con base en lo presentado en la Figura 10, e iniciando con la descripción de las soluciones específicas para los nodos, este nodo se encarga de realizar el procesamiento previo de las imágenes obtenidas por la cámara CSI. El objetivo es distribuir la imagen entre dos nodos diferentes y mejorar el rendimiento en el procesamiento, evitando problemas cuando dos nodos se conectan al mismo tópico.

- Su funcionalidad principal se activa cuando se realiza una suscripción al tópico /video_source/raw, que recibe imágenes crudas con el tipo de mensaje sensor_msgs/Image. Esta suscripción obtiene las imágenes de la cámara en formato BGR.
- Cada imagen pasa por un procesamiento básico que inicia con una rotación de 180° (para corregir la orientación de la cámara) y un redimensionamiento a 1200 x 480 píxeles.
- Para la salida destinada al tópico /filtered_image, que será utilizada por el nodo line_tracker, se realiza un recorte del área de interés, seleccionando una región comprendida entre los 230 y 480 píxeles de altura, y entre los 200 y 1000 píxeles de ancho.
- Posteriormente, la imagen se convierte a escala de grises y se aplica una umbralización fija con un valor de 110, seguida de una inversión binaria, de modo que la línea negra pase a representarse en blanco.
- Para limpiar el ruido, se aplica una operación de apertura morfológica con un kernel de 3x3 y 2 iteraciones.
- Aplicamos una apertura morfológica de kernel 3x3 con 2 iteraciones para limpiar el ruido.

- En cuanto al procesamiento para el tópico `/compressed_image`, esta imagen está destinada al nodo `detection_node`, el cual se encarga de la detección de objetos mediante un modelo previamente entrenado.
- El procesamiento inicia con un redimensionamiento de la imagen a 640 x 480 píxeles, con el objetivo de optimizar su rendimiento y evitar demoras durante la detección.
- Las imágenes se publican a una frecuencia de 30 Hz, se comprimen en formato JPEG y son enviadas al tópico correspondiente.

Nodo de seguidor de línea (line_tracker):

Este nodo se presenta para hacer el procesamiento de imagen, se encarga de poder detectar una línea central en las imágenes de la cámara CSI para poder calcular el error de alineación respecto al centro de nuestra imagen y con eso poder publicar nuestro error para el nodo de “controllerf”.

- Primero crearemos un suscriptor en el tópico `/filtered_image`, que transmite las imágenes de la cámara CSI, detectadas por el `camara_node`.
- Creamos dos publicadores, el primero de estos es el `/error` que se encargará de enviar el mensaje al nodo `controller`, el segundo de estos creará un `/debug_image` que hará que podamos ver como se muestra el resultado de nuestro análisis
- Colocamos un suscriptor llamado `/reset_line`, permitiendo reiniciar el estado interno de los centroides.
- Establecemos variables que serán para nuestro centroide y un vector para guardar los últimos valores de los tres centroides.
- Las imágenes recibidas serán convertidas en formato BGR mediante Bridge. Para poder superponer marcas visuales
- Detectamos los contornos en la imagen binaria, seleccionando el de mayor área superando los 500 píxeles
- Dicha segmentación se divide en 3 secciones horizontales, para que cada sección tenga un cálculo de su centroide.
- Para realizar el cálculo del error se usa el centroide 2 como referencia principal, usando un fórmula como:

$$\text{error} = \frac{\text{centroide 2 en } x - \text{centro de la imagen}}{\text{centro de la imagen}} \quad (11)$$

- El error se normaliza entre -1 a 1, los valores negativos indicando su centroide en dirección del lado izquierdo y en positivo marcando en lado derecho.
- Si la detección de centroides, muestra que el centroide 3, que se encuentra en la parte superior del video, desaparece cerca del centro de la imagen, con un umbral de 20% de la imagen respecto al centro hará que nuestro nodo entre en estado `recover`.
- En el estado `recover` se publicará en el tópico de `/error` un valor especial de 2.0 y activará dicho nodo en el `controller`.
- Las posiciones anteriores de los centroides son reiniciadas para evitar algún cambio brusco mientras el puzzlebot se detuvo, evitando falsos mientras iniciaba el modo `recover`.

- En modo recover, se usa el centroide 2 o 1 (el que esté disponible) para seguir estimando el error. Si el error baja de 0.05 tanto en negativo como positivo, se considera que el puzzlebot ha sido centrado con respecto en la línea y desactivando el modo recover.
- Para el seguimiento dentro de las curvas se establece una gestión de estabilidad visual, si todo los centroides están presentes, se verifica si el cambio según el centroide entre cuadros es abrupto, esto lo identifica con un umbral de más de 90 píxeles.
- Si el cambio es muy grande, se reutilizan los centroides del cuadro anterior para evitar saltos, o detección de otros objetos.

Nodo de detección implementado el modelo

Primero se crea una clase principal llamada “YoloDetectorNode”, la cual hereda de “Node” de ROS 2. Esta clase representa el nodo encargado de realizar la inferencia con YOLOv8 para detectar señales de tránsito y colores de semáforo a partir de una imagen comprimida. Dentro de esta clase se inicializan los siguientes componentes importantes:

- Se carga el modelo YOLOv8 personalizado con los pesos entrenados (archivo .pt), utilizando la biblioteca “Ultralytics”
- Se definen los diccionarios de clases personalizadas, con claves numéricas y nombres de clases, así como los colores representativos (por ejemplo, “rojo”, “verde”, “stop”, “give way”).
- Se configura una persistencia mínima de detección para evitar falsos positivos, obligando a que una clase esté presente por al menos 0.5 segundos antes de considerarla válida.
- Se inicializan las suscripciones y publicaciones ROS 2, escuchando el tópico de imágenes comprimidas (/compres_image) y publicando en dos tópicos distintos: uno para señales (/sign) y otro para colores(/color), ambos como enteros.
- Luego se define el método “image_callback” el cual se ejecuta cada vez que se recibe una imagen, codificando la imagen desde su formato comprimido
- Se realiza inferencia con el modelo YOLOv8, obteniendo las predicciones.
- Se filtran las detecciones según un área mínima configurable por clase para evitar detecciones pequeñas no significativas.
- Se verifica si una detección ha sido constante durante el tiempo necesario para ser considerada válida, utilizando una estructura de temporizador por clase.
- Se identifican las detecciones válidas por tipo (color o señal) y se publican en los tópicos correspondientes usando “Int32”.
- Se anotan visualmente las detecciones sobre la imagen con rectángulos y etiquetas, y se guarda cada frame en un video para futura revisión.
- Se imprimen mensajes informativos en consola con íconos personalizados según la clase detectada.
- Finalmente, se implementa el método “destroy_node” el cual se encarga de liberar los recursos utilizados para guardar el video cuando se destruye el nodo, asegurando un cierre limpio del archivo “.avi”.

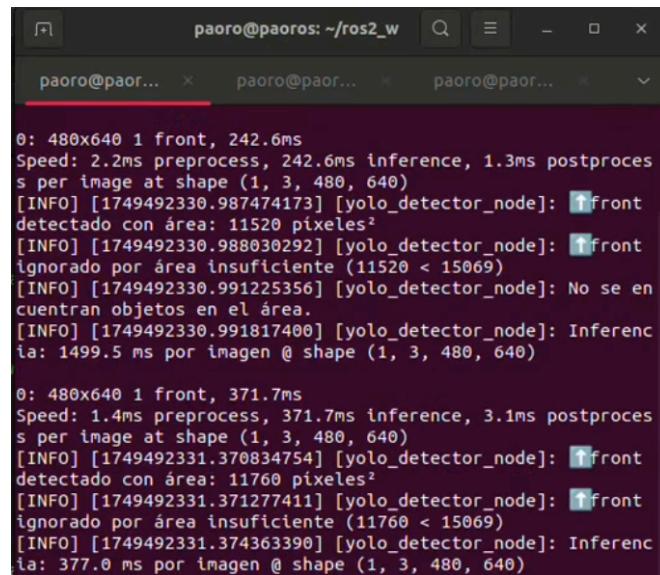
Controlador P (controllerf.py):

Este nodo se encarga de implementar un control para un robot móvil que responde al estado de la detección de objeto y también al seguimiento de línea anteriormente implementados, también se le atribuye el calcular la velocidad para navegar de forma autónoma, incluyendo comportamientos especiales como giros y recuperación tras pérdida de línea.

- Los tópicos establecidos son para, el error proporcional del seguimiento de la línea encargado por /error, el estado del semáforo por el tópico /color y las señales de tráfico detectadas son enviados por /sign.
- Establece salidas como /cmd_vel que son las velocidades lineal y angular para el robot. Y también /reset_line que inicia el seguimiento de línea tras completar una maniobra o recuperar la línea.
- Y establecemos una lista de variables y parámetros:
 - **Kp:** Ganancia proporcional para control de giro.
 - **base_speed:** Velocidad lineal básica.
 - **angular_speed:** Velocidad angular usada en giros.
 - **giro_angle_base:** Ángulo base del giro (80° en radianes).
 - **motion_state:** Estado del robot (STOP, SLOW, GO).
 - **active_signal:** Señal de tráfico actual activa.
 - **recovering:** Indica si el robot está en modo recuperación.
 - **recover_completed:** Marca si ya se ejecutó un RECOVER completo.
- En un modo de seguimiento normal se calcula el error cuando está dentro de los límites y no hay señales especiales, el cálculo cmd_vel proporcional al error y al estado del semáforo
- En la detección de un semáforo se aclaran las variables de STOP para detener el robot completamente, SLOW para reducir su velocidad y GO para moverlo normalmente
- Cuando entran las señales de tráfico se proponen las siguientes señales y sus respectivos movimientos:
 - Ahead: Avanza un tiempo determinado sin girar.
 - Stop: Detiene completamente el robot, evaluando el estado actual.
 - Give_way y work: Avanza lento mientras sigue la línea.
 - Turn_left y turn_right: ejecuta un movimiento por 3 fases que sería un avance inicial, giro de 90° dependiendo la dirección y avance final para retomar la línea.
- Si el tópico /error lanza un 2.0 inicia su modo recover, que busca emparejar el puzzlebot para poder realizar un movimiento con mejor precisión, deteniendo su movimiento cambiando la velocidad lineal a cero
- Mientras sigue en modo recover se corrige con el error recibido hasta que su error sea menor de 0.05 al igual que el negativo
- Después de concluir su modo recover espera la señal de turn_left, turn_right o ahead para continuar, ignorando los errores 2.0 hasta terminar los movimientos.

Resultados

Tras proponer la solución al problema, se pueden apreciar diferentes resultados de las partes que conforman este proyecto, dichos resultados van desde las terminales, las grabaciones de las perspectivas del nodo de line_follower y detection y una perspectiva del movimiento del robot dentro de la pista, en este apartado se documentaron los resultados obtenidos.



```
paoro@paoro: ~/ros2_w
```

```
0: 480x640 1 front, 242.6ms
Speed: 2.2ms preprocess, 242.6ms inference, 1.3ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1749492330.987474173] [yolo_detector_node]: front detectado con área: 11520 pixeles2
[INFO] [1749492330.988030292] [yolo_detector_node]: front ignorado por área insuficiente (11520 < 15069)
[INFO] [1749492330.991225356] [yolo_detector_node]: No se encuentran objetos en el área.
[INFO] [1749492330.991817400] [yolo_detector_node]: Inferencia: 1499.5 ms por imagen @ shape (1, 3, 480, 640)

0: 480x640 1 front, 371.7ms
Speed: 1.4ms preprocess, 371.7ms inference, 3.1ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1749492331.370834754] [yolo_detector_node]: front detectado con área: 11760 pixeles2
[INFO] [1749492331.371277411] [yolo_detector_node]: front ignorado por área insuficiente (11760 < 15069)
[INFO] [1749492331.374363390] [yolo_detector_node]: Inferencia: 377.0 ms por imagen @ shape (1, 3, 480, 640)
```

Figura 11: Terminal nodo detection

Como podemos observar en la terminal dentro de la computadora se ejecuta el programa de Detection_node que se encarga de buscar las señales definidas por el modelo best.pt como se aprecia en la Figura 11 el nodo busca un objeto y si vemos el front puede ser detectado, pero es ignorado debido a que no cumple la cantidad de pixeles establecidos.

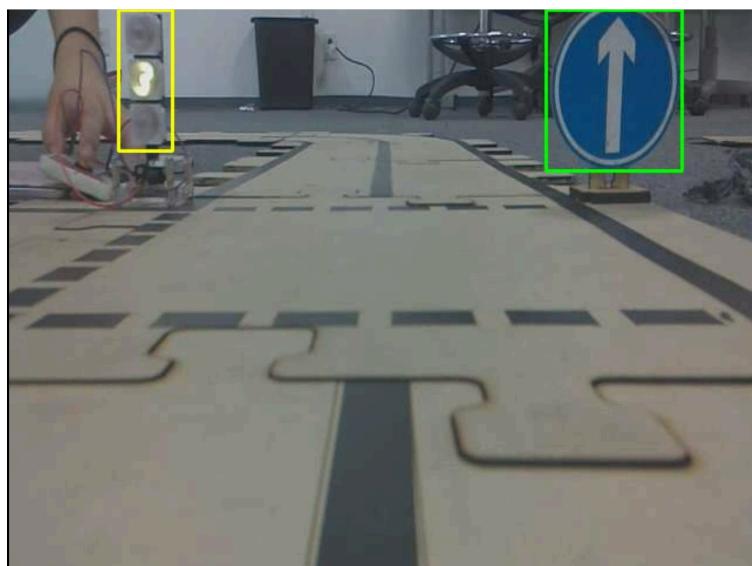


Figura 12: Video grabado por el nodo detection

La Figura 12 es la detección de objetos grabada desde el nodo de detection, como observamos está en una línea punteada significando que no se encontraran más linea para seguir, pero en las esquinas encontramos 2 objetos el semáforo en amarillo y el front, ambos estan en su bounding box que muestra que es lo que detectan en su imagen.

```
[controllerf-3] [INFO] [1748217979.937684327] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217980.037570264] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217980.136468441] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217980.236388181] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217980.343738962] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217980.436666931] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217980.5400559275] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217980.641224014] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217980.736736202] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217980.836743806] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217980.936807556] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217981.036955941] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217981.137657243] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217981.239618701] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217981.338840785] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217981.452328024] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217981.536082816] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217981.636992451] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217981.741112555] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217981.838823701] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217981.945793857] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217982.037532868] [controllerf]: Ejecutando STOP
[controllerf-3] [INFO] [1748217982.138557659] [controllerf]: Ejecutando STOP
```

Figura 13: Terminal launcher

En la terminal que se muestra como el launch engloba la parte de camara_node, line_follower y dection_node , en la Figura 13, es un momento dónde está ejecutando stop, significado que en el detection ha de haber encontrado la señal de stop y su tópico le ha informado a el nodo controller

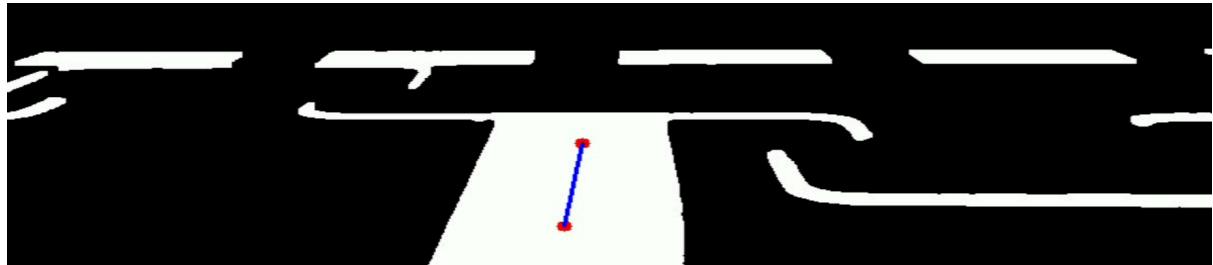


Figura 14: Video grabado por line_follower

Así es la perspectiva de line_follower donde en la vista de la Figura 14 se presenta como los filtros de camara_node han sido aplicados y por los centroides al notar la falta de un tercer centroide nos deja la interpretación que se encuentra dentro de un modo recover, con esto vemos que objetos externos no afecta el seguidor de línea.

Video demostrativo de la implementación de la solución.

En el siguiente enlace https://youtu.be/6oh_73FGgI se encuentra un video demostrativo sobre cómo lo antes mencionado se implementa en tiempo real, dándonos como resultado el recorrido completo del recorrido en el que obedece a las señales tanto del semáforo como de las señales viales, y haciendo el recorrido con el nodo de seguidor de línea para finalizar en donde inició.

Conclusiones

Por último, se presentan los logros alcanzados a lo largo del desarrollo del reto así como las reflexiones personales de cada uno de los integrantes del equipo. Se evalúa si los objetivos fueron alcanzados en su totalidad, identificando las razones detrás de su éxito o, en caso contrario, los factores que pudieron haber limitado su cumplimiento. Además de los aprendizajes y experiencias alcanzados durante la materia de implementación de robótica inteligente.

I. Conclusiones generales como equipo.

El desarrollo del sistema autónomo de navegación para el Puzzlebot representó una integración exitosa de los conocimientos adquiridos en percepción, control, aprendizaje automático y navegación móvil. A lo largo del reto, se diseñó, implementó y validó un sistema robótico funcional capaz de recorrer de manera autónoma una pista con trayectoria tipo “b”, siguiendo una línea negra, detectando semáforos y reaccionando correctamente ante señales de tránsito previamente entrenadas mediante visión por computadora.

En términos generales, los objetivos del proyecto se cumplieron satisfactoriamente. Se implementó un algoritmo de detección visual utilizando una red YOLOv8 entrenada con un conjunto de datos personalizado, capaz de identificar múltiples señales relevantes con alta precisión y confiabilidad. Además, se diseñó un sistema de seguimiento de línea robusto, que incluyó lógica de recuperación ante pérdida de la referencia visual, lo que permitió al robot mantener su curso incluso en presencia de interrupciones parciales.

El nodo de control integró de forma efectiva una máquina de estados, un controlador proporcional y una lógica jerárquica que respondió de forma coherente a múltiples condiciones del entorno: presencia de señales, estados del semáforo y errores de seguimiento. El sistema fue validado con pruebas reales, demostrando que el Puzzlebot puede navegar de forma completamente autónoma, sin intervención humana.

No obstante, algunos desafíos surgieron durante la ejecución, particularmente en la sincronización entre detección de señales visuales y la ejecución de las acciones correspondientes. En ciertos casos, la distancia entre la señal y el punto de reacción era limitada, lo cual requería de decisiones rápidas por parte del sistema. Aunque se logró mitigar este problema mediante umbrales de área mínima y validación por persistencia temporal, una mejora podría consistir en incorporar planeación anticipada basada en predicción de trayectorias o fusión de sensores.

Finalmente, este reto permitió una comprensión profunda de los procesos involucrados en el diseño de un sistema robótico cognitivo, desde la percepción y el control hasta la arquitectura de comunicación y el aprendizaje automático. Se desarrollaron competencias clave en integración de sistemas ROS 2, procesamiento visual en tiempo real y validación funcional de arquitecturas distribuidas. Las metodologías implementadas resultaron efectivas y

extensibles, y representan una base sólida para aplicaciones más complejas de robótica móvil autónoma.

II. Reflexión personal

Daniel Castillo López

Como conclusión, enfatizo que se ha logrado de manera efectiva el planteamiento para solucionar la navegación automática de nuestro Puzzlebot, debido a que se integraron los elementos necesarios para realizar la detección de línea, señales y semáforos. Esto permitió que los distintos conceptos interactúan de manera más eficiente dentro del entorno ROS, respetando las necesidades específicas de cada nodo. En cuanto al hardware, aunque se presentaron dificultades durante el desarrollo, como fallos en los motores o el posicionamiento de la cámara, estos no representaron un problema significativo, salvo en algunos casos donde la cámara enviaba un frame en negro, lo que complicaba la toma de decisiones en momentos clave, como al aproximarse a una curva.

Cabe señalar que este proyecto puede presentar ciertos errores en diferentes ambientes, especialmente dependiendo del nivel de reflejo de luz sobre la línea negra o cuando una fuente luminosa muy intensa interfiere con la visión computacional. Para mitigar estos problemas, se aplicó la robustez necesaria en el procesamiento de imagen, lo cual permitió operar de manera confiable en condiciones comunes. No obstante, recomiendo una mejora en la base de datos utilizada para el entrenamiento de la red neuronal, ya que fue construida principalmente con condiciones específicas del laboratorio, lo que puede limitar su desempeño ante cambios extremos de iluminación o entorno.

En términos de la arquitectura basada en suscriptores y temporizadores, se permite una reacción en tiempo real, con una lógica jerárquica que prioriza los comportamientos críticos. Esto es especialmente relevante en el caso específico de los giros y su detección, demostrando una estrategia de control orientada a tareas concretas, utilizando únicamente la perspectiva de la cámara. Además, el hecho de emplear un robot diferencial facilita el manejo y control con precisión. Con base en lo anterior, considero que el proyecto cumple con indicadores funcionales claros, lo que permitió construir una solución integral, operativa y alineada con los objetivos y necesidades del proyecto.

Emmanuel Lechuga Arreola

En cuanto a esta materia pude conocer muchos conceptos, así como la forma en la que ROS 2 funciona con hardware de robótica y cómo es que este puede llegar a tener una gran comunicación gracias a su manera de programar e implementar, por supuesto que el reto me ayudó a conocer más acerca de la robótica en sí como una materia multidisciplinaria, en la cual podemos aplicar diferentes conocimientos para alcanzar un objetivo en común que en este caso fue un robot autónomo el cual como lo dice su nombre tuviera una autonomía sin necesidad de intervención. y para desarrollar eso mismo tuvimos que ir dando pequeños pasos en el desarrollo de la movilidad ya que al inicio fue obtener las velocidades máximas y mínimas del puzzlebot, después ocupar la odometría, para después pasar con cumplir ciertos

aspectos de la misma, como una cierta distancia, después el sistema de visión donde tuvimos que incorporar procesamiento de imagen/video con el fin de primero identificar los colores, y en base a esto poder avanzar, después fue implementar el seguidor de línea y adicionarle el semáforo después, y al final tuvimos que hacer un modelo de redes neuronales el cual por medio de los procedimientos antes mencionados a lo largo del documento, el robot pudiera identificar lo que son las señales, el semáforo y la línea para poder generar un control en los motores y de esta manera, entregar un proyecto como el que entregamos donde el robot puede moverse a lo largo de esta pista en forma de b de manera autónoma, sin necesidad de intervención del equipo. por supuesto que aprendimos más que solo lo aplicado en el puzzlebot, por ejemplo las transformaciones de los brazos robóticos, los distintos modelos de entrenamiento, el para qué sirven las épocas en el mismo, así como las redes bayesianas que en su momento algunas veces dieron un poco de dolores de cabeza de manera general con todos los temas, debido a que es una materia en la cual podemos implementar todo lo antes visto así en los otros semestres, sobre todo con la parte de control de PID. Es algo tan gratificante ver como los conocimientos previos le dan forma a lo que hoy podemos llamar un sistema autónomo en un puzzlebot. y por eso mismo le doy gracias a mi equipo y a los profesores que me han acompañado a lo largo de la carrera para completar mi formación como profesionista y que espero seguir en comunicación con todos.

Paola Rojas Domínguez

Gracias a la materia de Implementación de Robótica Inteligente, tuve la oportunidad de trabajar en este proyecto integral que me permitió comprender de manera aplicada los distintos componentes que conforman un sistema autónomo para robots. El trabajo con el Puzzlebot resultó ser una práctica valiosa para entender el funcionamiento de un robot móvil diferencial, desde sus características físicas y cinemáticas hasta su interacción con el entorno mediante sensores y actuadores.

Este reto representó los primeros acercamientos con ROS 2, cuya arquitectura basada en nodos y comunicación es ampliamente utilizada en el mundo profesional. Aprendí a estructurar proyectos modulares, implementar tópicos de comunicación y depurar procesos en tiempo real dentro de un entorno robótico real.

Uno de los aspectos más enriquecedores fue integrar en un solo sistema componentes de cinemática, control, visión por computadora y redes neuronales. El reto me permitió conocer todo el proceso para crear un sistema autónomo el proceso desde tomar fotos para la base de datos, procesar imágenes, implementar el controlador P hasta corregir errores para volver el sistema robusto. El hecho de combinar algoritmos clásicos de procesamiento de imagen con modelos de aprendizaje profundo para interpretar señales visuales en tiempo real me permitió entender cómo se construyen sistemas cognitivos robustos que reaccionan a su entorno y lo complejo que es crearlos.

Este proyecto sirvió como primeros pasos hacia la creación de vehículos autónomos, abarcando no solo el control básico de movimiento, sino también la interpretación del entorno como las señales de tránsito y la adaptación del comportamiento del robot.

En conclusión, haber desarrollado un sistema de este estilo me dio una idea más clara de los retos que enfrenta la robótica móvil moderna y conocer temas relacionados con percepción, navegación y toma de decisiones autónoma. Por último me gustaría agradecer a nuestros profesores por su apoyo durante todo el semestre.

Anexos

Video explicativo: https://youtu.be/6oh_73FGgI

Video demostrativo: https://youtu.be/f_6fwlfva1s

Repositorio de github:

https://github.com/PaolaRojas24/robonautas/tree/main/Retos_Manchester_Robotics_IRI/Week10

Documento tecnico de nodos:

<https://drive.google.com/file/d/1R3XBfi0ysNlaHKm2mTL0In6CAdjIzaXM/view?usp=sharing>

Presentación:

https://www.canva.com/design/DAGpvUgY_CM/MK6Qin9KZn8HpOPRFJ0RGw/edit?utm_content=DAGpvUgY_CM&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Referencias

En este apartado se anexan los elementos consultados para el desarrollo del tema de investigación.

ManchesterRoboticsLtd. (s. f.-c).

*TE3002B_Intelligent_Robotics_Implementation_2025/Week1/Presentations/PDF/MC
R2_Puzzlebot_Jetson_Ed_ROS2.pdf at main · ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025.*

GitHub.

https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025/blob/main/Week1/Presentations/PDF/MCR2_Puzzlebot_Jetson_Ed_ROS2.pdf

ManchesterRoboticsLtd. (s. f.-b).

*TE3002B_Intelligent_Robotics_Implementation_2025/Week1/Presentations/PDF/MC
R2_Puzzlebot_Introduction_V2.pdf at main · ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025.*

GitHub.

https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025/blob/main/Week1/Presentations/PDF/MCR2_Puzzlebot_Introduction_V2.pdf

ManchesterRoboticsLtd. (s. f.-g).

*TE3002B_Intelligent_Robotics_Implementation_2025/Week3/Presentations/PDF/MC
R2_Closed_Loop_Control_v3.pdf at main · ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025.*

GitHub.

https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025/blob/main/Week3/Presentations/PDF/MCR2_Closed_Loop_Control_v3.pdf

Freddy. (s. f.). *Implementacion-de-robotica-inteligente-2025/Implementación de Robótica Inteligente (sesion 5).pptx* at *main · freddy-7/Implementacion-de-robotica-inteligente-2025.* GitHub.
[https://github.com/freddy-7/Implementacion-de-robotica-inteligente-2025/blob/main/Implementaci%C3%B3n%20de%20Rob%C3%ADtica%20Inteligente%20\(sesion%205\).pptx](https://github.com/freddy-7/Implementacion-de-robotica-inteligente-2025/blob/main/Implementaci%C3%B3n%20de%20Rob%C3%ADtica%20Inteligente%20(sesion%205).pptx)

Castillo, D., Lechuga, E., & Rojas, P. (s. f.). *Pose y Velocidades Angulares [Diapositivas].*
https://www.canva.com/design/DAGj6Fr_yI/IaSBza8tKyvkQKDbTiAbvQ/edit

ManchesterRoboticsLtd. (s. f.-j.).
TE3002B_Intelligent_Robotics_Implementation_2025/Week4/Presentations/PDF at *main · ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025.* GitHub.

https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025/tree/main/Week4/Presentations/PDF

ManchesterRoboticsLtd. (s. f.-k.).
TE3002B_Intelligent_Robotics_Implementation_2025/Week6/Presentations/PDF/MC R2_shape_detection_watermark.pdf at *main · ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025.* GitHub.

https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025/blob/main/Week6/Presentations/PDF/MCR2_shape_detection_watermark.pdf

ManchesterRoboticsLtd. (s. f.-l).

TE3002B_Intelligent_Robotics_Implementation_2025/Week7/Presetations/7-1

MCR2_AI.pdf at *main*

ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025.

GitHub.

https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025/blob/main/Week7/Presentations/7-1%20MCR2_AI.pdf