

RETO FINAL. Manchester Robotics

Abraham Ortiz Castro

Dpto. de Ingenierías Tec de Monterrey
Tecnológico de Monterrey
Puebla, México
A01736196@tec.mx

Alan Iván Flores Juárez

Dpto. de Ingenierías Tec de Monterrey
Tecnológico de Monterrey
Puebla, México
a01736001@tec.mx

Jesús Alejandro Gómez Bautista

Dpto. de Ingenierías Tec de Monterrey
Tecnológico de Monterrey
Puebla, México
A01736171@tec.mx

Ulises Hernández Hernández

Dpto. de Ingenierías Tec de Monterrey
Tecnológico de Monterrey
Puebla, México
A01735823@tec.mx

CONTENTS

I Resumen	2	V-B	Control de lazo cerrado para un robot móvil	4
II Subcompetencias a evaluar en el Reto.	2	V-C	PID aplicado a un robot móvil diferencial	5
III UNIDADES FUNCIONALES - Características del Robot	2	V-D	Robustez de un controlador	5
III-A	2	VI UNIDADES FUNCIONALES - Navegación	5	
Puzzlebot	2	VI-A	Odometría de un robot móvil diferencial	5
III-A1 ¿Qué es?	2	VI-B	Cálculo de la distancia y ángulo recorrido por un Robot Móvil	6
III-A2 Elementos	2	VI-C	Cálculo del error	6
III-A3 Interconexión	3	VII UNIDADES FUNCIONALES - Visión Computacional	6	
III-B	3	VII-A	Procesamiento de imágenes en la Jetson Nano	6
Comunicación SSH	3	VII-A1	OpenCV	6
III-B1 Configuraciones previas a la Jetson	3	VII-A2	CUDA	6
III-B2 Configuraciones para la comunicación	3	VII-B	Interconexión entre la Jetson y la cámara	6
IV UNIDADES FUNCIONALES - Cinemática	3	VII-B1	Cámara CSI:	6
IV-A	3	VII-B2	Cámara USB:	7
Teleop Twist Keyboard	3	VII-C	Detección de contornos o formas	7
IV-A1 Instalación y ejecución	3	VII-D	Detección de colores	7
IV-A2 Controles básicos del robot	3	VII-E	Robustez en sistemas de procesamiento de imágenes	7
IV-A3 Modo Holonómico	4	VII-F	Seguidor de Línea con cámara	7
IV-B	4	VII-F1	Consideraciones	7
Mensajes Twist	4	VII-G	Robustez de un Seguidor de Línea	8
IV-B1 ¿Qué significan X, Y y Z en los mensajes Twist?	4	VII-H	Algoritmos implementados para el seguimiento de líneas	8
IV-B2 Creación de mensajes tipo Twist	4	VIII UNIDADES FUNCIONALES - Aprendizaje Profundo	8	
IV-C	4	VIII-A	Redes Neuronales Convolucionales (CNN)	8
Cinemática de un Robot Diferencial	4	VIII-A1	Código de ejemplo de una CNN en Python	8
IV-C1 Principio de funcionamiento de un robot diferencial:	4	VIII-A2	Entrenamiento de red YOLO (You Only Look Once)	8
IV-C2 Modelo cinemático:	4	VIII-A3	Roboflow	9
IV-C3 Uso de modelos cinemáticos:	4			
V UNIDADES FUNCIONALES - Control	4			
V-A	4			
Lazo abierto de control en un Robot Móvil	4			
V-A1 Definición:	4			
V-A2 Usos	4			
V-A3 Ventajas y Desventajas	4			

IX	Solución del problema	9
IX-1	Nodos Laptop	9
IX-2	Nodos Jetson Nano	11
X	Resultados	13
X-A	Interpretación de resultados	13
X-B	Robustez de implementación	14
XI	Conclusiones	14
XII	Reflexiones personales	15
XII-1	Alejandro Gómez	15
XII-2	Abraham Ortiz	15
XII-3	Alan Flores	16
XII-4	Ulises Hernández	16
References		16
Appendix A: Repositorio del Proyecto		17
Appendix B: Presentación ManchesterRobotics		17
Appendix C: Video Demostrativo		17

I. RESUMEN

El presente documento detalla la implementación de la solución al desafío final planteado por Manchester Robotics en la segunda semana del segundo periodo. A lo largo de las tres semanas posteriores, se desarrolló, innovó y mejoró una solución integral que combina los desafíos previos (identificación de señales, seguimiento de líneas y redes neuronales convolucionales) en el PuzzleBot. El objetivo final fue entregar un proyecto de robótica inteligente capaz de seguir un recorrido predeterminado (una pista en forma de P) utilizando un seguidor de línea basado en visión computacional. Además, se incorporó la identificación de señales de tránsito (*Roundabout, Ahead Only, Turn Right, Turn Left, Road Work, Give Way, Stop, Red Light, Yellow Light y Green Light*) y la lógica necesaria para interactuar adecuadamente una vez identificadas por la cámara.

Este proyecto implicó un trabajo exhaustivo en el diseño de algoritmos para la identificación de señales y el seguimiento de líneas, enfrentando y superando las limitaciones del entorno de trabajo (Jetson Nano). Se desarrollaron soluciones robustas y eficientes para manejar estas y otras interacciones en un entorno dinámico y en tiempo real.

II. SUBCOMPETENCIAS A EVALUAR EN EL RETO.

- Programa de forma óptima unidades funcionales que permiten a un sistema electrónico interactuar, de modo efectivo, con el medio; generando de manera idónea la percepción y manipulación con el entorno.
- Implementa de manera eficiente la interacción funcional entre la unidad de procesamiento y sus periféricos (internos y externos) para dar solución precisa a la necesidad

planteada, a partir de utilizar correctamente las herramientas de diseño.

- Construye de manera integral soluciones para atender estratégicamente problemas y necesidades sociales, promoviendo en forma efectiva la colaboración entre actores, y la generación de acuerdos sostenibles, solidarios y congruentes con los Objetivos de Desarrollo Sostenible (ODS) o con otros marcos de referencia que fortalecen la democracia y el bien común.
- Transmite de forma óptima mensajes escritos en los que expresa ideas con precisión y las presenta de manera estratégica para hacerlas comprensibles a un destinatario o público; al emplear un lenguaje adecuado al propósito y situación comunicativa; utilizar de modo idóneo los recursos paratextuales pertinentes (negritas, itálicas, subrayado, tipología, tamaño de letra, etc.) y, al estructurar de manera articulada los elementos requeridos por el género discursivo.

III. UNIDADES FUNCIONALES - CARACTERÍSTICAS DEL ROBOT

A. Puzzlebot

1) *¿Qué es?:* El PuzzleBot, desarrollado por Manchester Robotics es un robot de laboratorio especializado en la educación y en el prototipado, siendo esa la misión del mismo, la del desarrollo de aprendizaje en la robótica a través de este bot. Existen algunas variantes del PuzzelBot que no son más que la misma estructura robótica, con implementaciones de algunas tarjetas o controladores adicionales que le permiten desarrollar nuevas complejidades de algoritmos, así como realizar tareas que los otros modelos no pueden realizar. Entre estas versiones podemos destacar las siguientes:

- PuzzleBot Hacker Edition.
- PuzzleBot Jetson Edition.

El PuzzleBot está compuesto por componentes fuera de serie primordialmente para lograr un precio bajo frente a un gran volumen de producción, con la intención de facilitar el acceso a este producto de desarrollo en robótica a una mayor cantidad de usuarios que adquieran el PuzzleBot completo o en su defecto las partes del mismo de forma independiente.

[1]

2) Elementos:

- **Hacker Board:** Actúa como un nodo computacional robusto para los componentes del kit, contando con una interfaz de comunicación con unidades externas.
- **NVIDIA Jetson Nano:** Es considerada como la “mente” del PuzzleBot, pues permite la implementación de algoritmos de aprendizaje profundo y visión por computadora.
- **Cámara Raspberry Pi:** Es empleada para la captura de imagen y video que junto al poder de la Jetson pueden procesar algoritmos de visión computacional.
- **Comunicación serial:** Tanto la Hacker Board como la Jetson Nano están comunicadas mediante comunicación serial.

3) *Interconexión:* El PuzzleBot Jetson Edition es un robot diseñado para que los usuarios implementen algoritmos de bajo nivel en tiempo real en la Hacker Board y algoritmos de alto nivel de como Inteligencia Artificial y visión por computadora en la NVIDIA Jetson Nano. La interconexión entre la Hacker Board y NVIDIA Jetson Nano se realiza a través de comunicación serial.

En el caso del PuzzleBot, la interconexión permite que los diferentes componentes propios del robot se comuniquen y trabajen juntos para realizar tareas complejas. [1]

B. Comunicación SSH

La comunicación SSH en ROS hace referencia a la configuración de múltiples dispositivos para la comunicación entre los mismos a través de ROS. Particularmente, para fines de la práctica nos interesa la comunicación con el robot PuzzleBot a través de SSH, a continuación se explica acerca de los procesos necesarios previos y en el momento de establecer comunicación entre la Jetson Nano y nuestra computadora. [2]

1) Configuraciones previas a la Jetson:

- *Instalación del Sistema Operativo:* Es necesario descargar el image del PuzzleBot e instalarlo en una tarjeta SD desde un computador externo, una vez este listo es necesario incertarla en el compartimento de la Jetson para su funcionamiento.
- *Configuración del Hotspot:* Es necesario emplear una tarjeta de red inalámbrica, ya que la Jetson no cuenta con una. Dentro de la interfaz de la Jetson desde el administrador de Red configuraremos el hotspot, por defecto contará con un nombre y contraseña predeterminado, pero este puede cambiarse. Guarda los cambios para que se apliquen a la Jetson. [2]

2) Configuraciones para la comunicación:

- *Conectar el dispositivo a la red del PuzzleBot:* Una vez el hotspot de la Jetson Nano este configurado, desde nuestra computadora podremos visualizarla como una red disponible, a la cual deberemos conectarnos. Si no configuramos la red de forma personalizada, necesitaremos ingresar la contraseña, la cual es por defecto **Puzzlebot72**.
- *Conectarse vía SSH:* Una vez conectada a la red de la Jetson, desde una ventana de terminal ingresamos el comando `ssh puzzlebot@10.42.0.1`, los últimos dígitos de la línea indican la IP por defecto, si deseas configurarla, recomendamos cambiar el último dígito (altamente recomendable al trabajar en el mismo espacio con otros equipos), después de ello, el comando solicitará una contraseña, la cual es la misma que la de la red, en este caso **Puzzlebot72**. [3]

IV. UNIDADES FUNCIONALES - CINEMÁTICA

A. Teleop Twist Keyboard

`teleop_twist_keyboard` se trata de un paquete propio de ROS el cual permite un modo de teleoperación de

teclado de manera genérica. Este nodo es el encargado de la traducción de los pulsos sobre las teclas en mensajes. Este paquete basa su funcionamiento en el tipo de mensaje `Twist` de ROS, los cuales no son más que la forma más común de representar velocidades en un espacio tridimensional. Cada uno de estos contiene componentes principales, y primordialmente contamos con dos de ellos:

- *Linear:* un vector tridimensional que representa las velocidades lineales existentes en las direcciones x, y, z.
- *Angular:* un vector tridimensional que representa las velocidades angulares existentes alrededor de los ejes x, y, z.

[4]

1) *Instalación y ejecución:* Para instalar el paquete es necesario aplicar el siguiente comando sobre la terminal:
`sudo apt- install ros install ros-noetic-teleop-twist-keyboard` Así mismo, para hacer uso del nodo, debemos llamarle mediante el comando en terminal:
`rosrun teleop_twist_keyboard teleop_twist_keyboard.py`

2) *Controles básicos del robot:* Una vez que el nodo se ejecute, podrás usar las teclas del teclado para el control del robot, a continuación se mencionan las entradas (teclas) básicas para el control del robot:

- **i:** activa los motores y con ello las ruedas del robot de forma en que estas giren hacia adelante.
- **,**: activa los motores y con ello las ruedas del robot de forma en que estas giren hacia atrás.
- **k:** detiene a los motores y con ello el movimiento del robot.
- **j:** activa ambos motores, uno lo mantiene el giro "hacia adelante" y el otro lo invierte, de forma que el robot se posiciona en dirección a la derecha.
- **l:** activa ambos motores, uno lo mantiene el giro "hacia adelante" y el otro lo invierte, de forma que el robot se posiciona en dirección a la izquierda.
- **u:** activa ambos motores, uno con una velocidad de giro mayor que otro, de forma que girará hacia la derecha - adelante de manera más amplia en comparación con j.
- **o:** activa ambos motores, uno con una velocidad de giro mayor que otro, de forma que girará hacia la izquierda - adelante de manera más amplia en comparación con l.
- **m:** activa ambos motores, uno con una velocidad de giro mayor que otro, de forma que girará hacia la derecha - atrás de manera más amplia en comparación con j.
- **:** activa ambos motores, uno con una velocidad de giro mayor que otro, de forma que girará hacia la izquierda - atrás de manera más amplia en comparación con l.
- **q/z:** aumenta (q) y disminuye (z) las velocidades **MÁXIMAS** en un 10%
- **w/x:** aumenta (w) y disminuye (x) la velocidad **LINEAL** en un 10%.
- **e/c:** aumenta (e) y disminuye (c) la velocidad **ANGULAR** en un 10%.
- **Cualquier otra tecla:** detenemos el robot.
- **CTRL C:** Salir de los controles

3) *Modo Holonómico*: Si el robot es de tipo holonómico, es decir, puede desplazarse de manera lateral, se mantiene presionando la tecla *shift* y empleando alguno de los controles básicos del robot. [4]

B. Mensajes Twist

El tipo de mensaje Twist en ROS son uno de los elementos más importantes para la comunicación entre nodos de un sistema robótico, primordialmente empleados para enviar comandos de velocidad a un robot.

El mensaje tipo Twist consta de dos componentes de velocidad en espacio libre, las cuales se dividen en un vector de tres dimensiones definidas de la siguiente manera:

- **Twist.linear:** Representando la velocidad lineal con sus componentes en x, y y z.
- **Twist.angular:** Representando la velocidad angular alrededor de x, y y z.

En un robot terrestre, es altamente probable que la velocidad lineal pueda cambiar en los componentes de x y de y, sin embargo, la velocidad angular se concentrará principalmente en el eje z.

1) *¿Qué significan X, Y y Z en los mensajes Twist?:* En el espacio existen 3 ejes (x, y, z) los cuales son mutuamente perpendiculares entre sí y su punto de intersección es el origen $x=0$, $y=0$ y $z=0$. Esto funge como un marco de referencia que nos permite definir varios puntos. Tanto la velocidad angular y la linear emplean el mismo marco de referencia. [5]

2) *Creación de mensajes tipo Twist:* Para crear un mensaje tipo twist en Python, se sigue la siguiente estructura:

```
import rospy
from geometry_msgs.msg import Twist

# Crear una instancia de rospy.Publisher()
pub = rospy.Publisher('turtle1/cmd_vel', Twist, queue_size=1)

# Crear una instancia de un mensaje Twist
twist = Twist()

# Asignar valores a las componentes lineales y angulares
twist.linear.x = 1.0
twist.angular.z = 1.0

# Publicar el mensaje Twist
pub.publish(twist)
```

[6]

C. Cinemática de un Robot Diferencial

Se entiende a la cinemática de un Robot Diferencial como el estudio del movimiento de este tipo de robots sin contemplar las fuerzas que originan este movimiento. Un robot diferencial es un tipo de vehículo terrestre que emplea dos ruedas independientes o lo que es igual a dos ruedas con su propio motor, provocando que su locomoción o movimiento se base en la diferencia de velocidades de sus dos ruedas sobre un único eje.

El propósito de la cinemática diferencial no es otro que el de encontrar las relaciones entre las velocidades de nuestro punto de control $h(x, y)$ y las velocidades que intervienen en el movimiento del robot, omitiendo las fuerzas que ejercen sobre el robot (como la inercia y el rozamiento). [7]

1) Principio de funcionamiento de un robot diferencial::

- Si dos ruedas giran en la misma dirección y velocidad, el robot se moverá en línea recta hacia adelante.
- Si cambia el sentido de giro y mantiene la velocidad, el vehículo se desplazará hacia atrás.
- Si dos ruedas giran en la misma dirección, pero con diferentes velocidades, el robot se alejará del motor más rápido, es decir, si la rueda derecha gira más que la otra, el vehículo se moverá a la izquierda.
- Si ambas ruedas giran a la misma velocidad, pero en direcciones opuestas, el vehículo girará en su propio eje en sentido horario o antihorario.

2) *Modelo cinemático::* Para determinar la localización del móvil en el plano cartesiano, se emplea el modelo cinemático diferencial directo, el cual relaciona las velocidades del punto de control con las velocidades de los actuadores. Para obtener el modelo, se guiará de la geometría del vehículo, cuya posición se define en torno al punto h y con respecto a la orientación del ángulo. [8]

3) *Uso de modelos cinemáticos::* En general, el uso de estos modelos cinemáticos nos permiten realizar análisis y diseños de algoritmos de control, particularmente para tareas de robótica donde existe una baja velocidad y una poca carga en la estructura. [9]

V. UNIDADES FUNCIONALES - CONTROL

A. Lazo abierto de control en un Robot Móvil

1) *Definición::* Un sistema de control de lazo abierto es aquel cuyas acciones de control son previamente programadas y sin realizar ajustes en función de una retroalimentación, como en lazo cerrado. Este tipo de control asume que tanto las entradas como las salidas del sistema se conocen y pueden ser predichas, es decir, el controlador envía señales de control sin conocer a ciencia cierta que el estado real del sistema sea este. [10]

2) *Usos:* Se utiliza en situaciones donde la precisión no es imprescindible o cuando la variable de salida es fácilmente predecible, podría decirse que este tipo de control es recomendable para tareas repetitivas. El controlador enviará una secuencia predefinida de movimientos sin considerar si los mismos están siendo ejecutados de manera correcta.

3) *Ventajas y Desventajas:* Una de las ventajas más importantes de un control de lazo abierto es la simplicidad del mismo, pues al necesitar un sistema de retroalimentación, es mucho más fácil de implementar. No obstante, es esta simplicidad también un arma de doble filo, pues no hay forma de corregir errores durante la ejecución del sistema de control. Por lo anterior, es necesario que el control de lazo abierto sea empleado en sistemas donde la retroalimentación no es necesaria y, por el contrario, genera un costo adicional frente a los beneficios que provee. [11]

B. Control de lazo cerrado para un robot móvil

El control de lazo cerrado o de retroalimentación, es un tipo de control en el que se utiliza la salida del sistema para ajustar

la entrada con el objetivo de mantener un sistema en un estado deseado.

Para un robot móvil diferencial, el control de lazo cerrado implica el uso de la retroalimentación para ajustar las velocidades de las ruedas del robot con el fin de mantenerlo en una trayectoria esperada o deseada. Este proceso implica lo siguiente:

- **Modelación del sistema:** Implica entender el cómo las velocidades de las ruedas del robot afectan el movimiento, a partir de las ecuaciones de cinemática que describen al robot.
- **Medición del estado:** Esto puede ser logrado a partir del uso de sensores como el encoder en las ruedas, los cuales nos permiten medir la rotación de las ruedas y con algunos datos más propios del robot diferencial, calcular la distancia recorrida del robot.
- **Cálculo del error:** Conociendo el estado actual del robot, se puede conocer el error de la posición y del ángulo al conocer la diferencia entre esos valores deseados y los valores reales o actuales que conocemos nos dará el error existente para poder lograrlo.
- **Ajuste de las velocidades de las ruedas:** A partir del error, ajustamos las velocidades de las ruedas del robot, para nuestros fines a través de un controlador PID, lo cual permitirá ajustar la velocidad de las ruedas en función del error

[12]

C. PID aplicado a un robot móvil diferencial

Un controlador PID (Proporcional Integrador Derivativo) es una técnica de control empleada para sistemas de lazo cerrado. Enfocado en un robot móvil diferencial, el PID tendrá la tarea de ajustar las velocidades de las ruedas del robot en función de la medición del error. Cada una de las siglas de PID describe el tipo de control que ejercerán sobre el sistema, estas acciones son:

- **Control Proporcional (P):** Basándose en el error actual del control, si el error medible es grande, el controlador de tipo P intentará corregirlo rápidamente al aumentar la velocidad de las ruedas al producir una salida proporcional al error actual. La constante de proporcionalidad determina cuánto cambio en la salida se produce por unidad de cambio del error.
- **Control Integral (I):** Este elemento del control se enfoca en los errores pasados, es decir, si el robot ha estado fuera de la trayectoria por un tiempo, el controlador de tipo I comenzará a acumular este error con el tiempo y lo solucionará al ajustar la velocidad de las ruedas, proporcionando así un control de acción lenta, pero constante, ayudando a eliminar el error residual que a menudo ocurre al usar el controlador de tipo P.
- **Control Derivativo (D):** Se basa en la tasa de cambio del error, en otras palabras, si el robot vuelve rápidamente a su trayectoria, el controlador de tipo D disminuirá la velocidad con el fin de evitar sobreajustes. Este control

proporciona una acción rápida y proporcional a la tasa de cambio del error, disminuyendo así la respuesta del sistema y evitando oscilaciones.

Si bien es cierto que el controlador PID puede ser muy efectivo dentro de un robot diferencial móvil, necesita de un ajuste cuidadoso de los parámetros P, I y D para un mejor rendimiento. Aunque los valores se pueden obtener mediante métodos heurísticos, podemos recurrir a técnicas más efectivas como el método Ziegler-Nichols.

Adicionalmente, a lo anterior, no debemos dejar de lado que el control PID asume un modelo lineal del sistema, por lo que ciertos comportamientos del robot móvil de tipo diferencial puede mostrar comportamientos no lineales como consecuencia del deslizamiento de las ruedas o la dinámica del robot. [13]

D. Robustez de un controlador

Se entiende a la robustez del controlador a la capacidad de mantener un rendimiento de forma adecuada pese a las variaciones o incertidumbres en el sistema donde se está ejerciendo el control, se puede entender por ende que un controlador robusto es aquel capaz de manejar variaciones en los parámetros del sistema con el fin de mantener los objetivos, su estabilidad y el rendimiento.

Particularmente en un sistema PID, la robustez hace referencia a la capacidad del controlador para manejar variaciones sin que su rendimiento se reduzca, es decir, pese a que el sistema experimente cambios en sus parámetros, un controlador robusto es capaz de adaptarse al cambio mientras continúa dando un control apropiado o mejor dicho, un control que menos sensible a cambios bruscos del sistema. [16]

VI. UNIDADES FUNCIONALES - NAVEGACIÓN

A. Odometría de un robot móvil diferencial

La odometría en un robot móvil de tipo diferencial se basa en la estimación de la posición así como de la orientación del vehículo a partir de la información de la rotación registrada por sus ruedas a través del tiempo.

Un robot diferencial cuenta con dos ruedas de tracción con movimiento independiente, lo cual le permite al robot moverse en cualquier dirección sin cambiar su orientación. Su movimiento a grandes rasgos se basa en dos estados: cuando sus ruedas se mueven a la misma velocidad, avanzarán o retrocederán, mientras que la diferencia de velocidades entre estas hará que el robot gire.

La distancia recorrida por rueda puede estimarse al multiplicar el número de rotaciones de la rueda por su circunferencia, además, si se conoce la distancia entre ambas ruedas (equivalente a la longitud de la base del robot) la diferencia de las distancias recorridas puede ayudarnos a encontrar su orientación. Para entender mejor la odometría del robot diferencial, a continuación se mencionan algunas de las ecuaciones para la odometría de un robot diferencial.

• Distancia total recorrida:

$$\Delta d = \frac{d_d + d_i}{2} \quad (1)$$

Se trata de la distancia promedio recorrida por las dos ruedas. Se basa en la suposición de que el robot se mueve a lo largo de la línea que biseca el ángulo formado por las direcciones de movimiento de las ruedas.

- **Cambio de orientación:**

$$\Delta\theta = \frac{d_d - d_i}{l} \quad (2)$$

Se trata del cambio de orientación del robot. Se basa en la suposición de que el propio robot gira alrededor de un punto en el centro de la longitud entre ambas ruedas.

- **Cambio de coordenadas en X:**

$$\Delta x = \Delta d \cdot \cos(\theta) \quad (3)$$

Cambio en la posición en el eje de las X del robot, calculada al multiplicar la distancia total recorrida por el coseno de la orientación actual del robot.

- **Cambio de coordenadas en Y:**

$$\Delta y = \Delta d \cdot \sin(\theta) \quad (4)$$

Cambio en la posición en el eje de las Y del robot, calculada al multiplicar la distancia total recorrida por el seno de la orientación actual del robot.

Estas ecuaciones, aunque funcionales, no toman en cuenta algunos factores externos (como el deslizamiento de la rueda, el tipo de superficie y demás) que podrían arrojar estimaciones poco precisas. [14]

B. Cálculo de la distancia y ángulo recorrido por un Robot Móvil

Para el cálculo de la distancia y el ángulo se utilizan las ecuaciones 5 y 6, las cuales utilizan la velocidad angular de cada uno de los motores (ω_L y ω_R), la distancia entre las ruedas (r) y la distancia entre los ejes (l).

$$d = r\left(\frac{\omega_R + \omega_L}{2}\right) * dt \quad (5)$$

$$\theta = r\left(\frac{\omega_R - \omega_L}{l}\right) * dt \quad (6)$$

Estos datos serán utilizados posteriormente para definir si se ha alcanzado las coordenadas preestablecidas en el nodo `path_generator`.

C. Cálculo del error

Para generar un sistema robusto de navegación y control del robot, el cálculo del error en un robot móvil de tipo diferencial es fundamental, y se basa fundamentalmente en la diferencia entre la posición actual del robot y la posición deseada u objetivo.

El error en un robot diferencial puede abordarse en dos componentes primordiales.

- **Error de posición:**

$$e_p = P_o - P_a \quad (7)$$

Es la diferencia entre la posición deseada y su posición actual, básicamente, es la diferencia entre las coordenadas de la posición objetivo y las actuales del robot.

- **Error de orientación:**

$$e_\theta = \theta_o - \theta_a \quad (8)$$

Es la diferencia entre la orientación deseada y su orientación actual, básicamente, es la diferencia entre el ángulo objetivo y actual del robot.

[15]

VII. UNIDADES FUNCIONALES - VISIÓN COMPUTACIONAL

A. Procesamiento de imágenes en la Jetson Nano

La Jetson Nano es una tarjeta embebida empleada ampliamente en el procesamiento de imágenes y en el desarrollo de visión computacional debido a sus capacidades de GPU y a su compatibilidad con bibliotecas diseñadas para el procesamiento de imágenes tales como OpenCV y CUDA. Aunado a lo anterior, este sistema embebido es capaz de ejecutar varias redes neuronales en paralelo diseñadas para algoritmos de clasificación de imágenes, detección de objetos, segmentación y el procesamiento de lenguaje. [17]

1) *OpenCV*: Open Source Computer Vision Library, mejor conocida como OpenCV es la biblioteca de código abierto más grande del mundo especializada en visión computacional, es por ello que comúnmente es empleada en una gran gama de aplicaciones. Además de realizar varias operaciones de procesamiento de imágenes, cuenta con una gran compatibilidad con muchos sistemas operativos (Android, iOS, Linux, Windows, etc.) y lenguajes de programación (Python, C++, Java, etc.) Algunas de las cosas que OpenCV permite al usuario son.

- Reconocimiento facial.
- Identificar objetos.
- Clasificar acciones humanas en videos.
- Seguir los movimientos de la cámara.
- Seguir objetos en movimiento.
- Extraer modelos 3D de objetos.
- Encontrar similitudes entre imágenes en una misma base de datos.
- Seguir el movimiento de los ojos.

[18]

2) *CUDA*: Compute Unified Device Architecture, o por sus siglas CUDA, es una plataforma de computación en paralelo desarrollada por NVIDIA, destinada a codificar algoritmos que se ejecutan en paralelo en la GPU, lo cual es especialmente útil a la hora de procesar imágenes en donde las operaciones pueden ejecutarse paralelamente a cada pixel. [19]

B. Interconexión entre la Jetson y la cámara

La Jetson Nano de NVIDIA ofrece dos opciones para conectar cámaras: a través de la interfaz CSI (Interfaz de Sensor de Cámara) o mediante USB.

1) *Cámara CSI*:: Para la conexión CSI, se debe realizar lo siguiente:

- Comprobar que la Jetson esté apagada y desconectada.
- Localizar el conector CSI en la placa.

- Levantar cuidadosamente la pestaña de plástico del conector.
- Deslizar lentamente el cable de la cámara.
- Vuelve a apretar la pestaña de plástico en su lugar cuidadosamente con el cable.

[20]

2) *Cámara USB*:: Solo bastará con conectar a uno de los puertos USB de la Jetson una cámara mediante USB.

Una vez se realizó la conexión, puedes emplear la biblioteca OpenCV (previamente vista) para capturar y procesar imágenes. [21]

C. Detección de contornos o formas

Detectar contornos o formas en una imagen es una tarea frecuente en el procesamiento de imágenes y la visión por computadora. OpenCV ofrece una variedad de funciones diseñadas para simplificar este proceso.

Aunque hay una gran variedad de maneras en las que se puede abordar esta tarea, generalmente se suele seguir una metodología concreta usando OpenCV, la cual es la siguiente:

- **Leer la imagen:** leemos la imagen que contiene las formas usando la función.
- **Convertir a escala de grises:** convertimos la imagen a escala de grises.
- **Aplicar umbral:** aplicamos un umbral a la imagen para obtener una imagen binaria.
- **Encontrar contornos:** encontrar los contornos externos de las formas en la imagen binaria.
- **Aproximar contornos a polígonos:** aproximar los contornos a polígonos con una precisión especificada.
- **Identificar formas:** usamos el número de vértices y la relación de aspecto de los polígonos para identificar las formas y etiquetarlas en la imagen. También podemos usar características propias como excentricidad o compactidad si se trata de detectar círculos o elipses.

[22]

D. Detección de colores

Detectar colores en imágenes es una tarea habitual en el procesamiento de imágenes y la visión por computadora. OpenCV, una biblioteca popular para tales tareas, ofrece funciones específicas para simplificar este proceso. Específicamente, el espacio de color HSV (Hue, Saturation, Value / Matiz, Saturación, Brillo) resulta especialmente útil para la detección de colores.

El espacio de color HSV (Matiz, Saturación, Valor) es preferido sobre RGB para la detección de colores debido a su capacidad para separar los componentes del color de manera más intuitiva, su invariancia al brillo, su eficiencia computacional y su robustez frente a cambios de iluminación.

Al igual que con la detección de formas, la detección de colores sigue un esquema principal para lograr dicha tarea, este proceso es:

- **Leer la imagen:** leemos la imagen a analizar.
- **Convertir a escala de colores HSV:** convertimos la imagen a escala de colores HSV.

- **Aplicar umbral:** definimos los rangos de colores que queremos detectar en la escala HSV y aplicamos un umbral a la imagen para obtener una imagen binaria.
- **Encontrar contornos:** encontrar los contornos de las formas en la imagen binaria a resaltar.
- **Dibujar contornos y mostrar imagen:** dibujamos los contornos de las formas detectadas en la imagen original y mostramos la imagen resultante.

[23]

E. Robustez en sistemas de procesamiento de imágenes

La robustez en los sistemas de procesamiento de imágenes se define como la capacidad del sistema para manejar variaciones y perturbaciones en las imágenes de entrada y, a pesar de ello, generar resultados precisos y consistentes.

Por lo general, estas variaciones y perturbaciones suelen ser causadas por varios factores, siendo los más comunes los siguientes:

- **Ruido:** Un sistema robusto debe ser capaz de manejar el ruido y producir resultados precisos, pese a sufrir complicaciones con la calidad de la cámara o la compresión de la imagen.
- **Variaciones de iluminación:** Un sistema robusto debe tener la capacidad de afrontar estas variaciones y producir resultados consistentes.
- **Deformaciones y variaciones de escala:** Un sistema robusto es capaz de reconocer formas y objetos, pese a estos mostrar variaciones en sus formas como consecuencia de la perspectiva o distancia de la cámara.
- **Variaciones en la orientación:** Un sistema robusto debe tener la capacidad de reconocer objetos sin importar su orientación.

Así mismo, un sistema robusto deberá ser capaz de manejar diferentes resoluciones y/o tamaño de imágenes y procesarlas en tiempo real o cercano a ello. [24]

F. Seguidor de Línea con cámara

Se entiende a un seguidor de línea con cámara es una forma avanzada de seguidor de línea con visión artificial, al capturar su entorno y procesarlo para identificar la línea de seguimiento.

1) Consideraciones:

- **Captura de imágenes:** El robot deberá contar con una cámara que capture el suelo mientras se desplaza, esta imagen se concentra en la línea y el fondo.
- **Procesamiento de imágenes:** Utiliza algoritmos de visión por computadora para detectar la línea. Algunos algoritmos empleados tradicionalmente implican la conversión de la imagen a escala de grises, la detección de bordes y la identificación de la línea en función de su color o contraste con el fondo.
- **Algoritmos de control:** Una vez que se detecta la línea, el robot ajusta su movimiento para seguir la trayectoria.
- **Corrección de la posición:** Si se desvía demasiado hacia un lado, el algoritmo de control ajusta la dirección para volver a la línea, todo esto en tiempo real.

[25]

G. Robustez de un Seguidor de Línea

Algunos aspectos a considerar para la robustez del robot de seguidor de línea con cámara son:

- **Detección de línea:** Para un algoritmo sólido, se necesitan técnicas avanzadas de procesamiento de imágenes, como la umbralización adaptativa, filtrado de ruido y corrección de distorsiones.
- **Tolerancia a variaciones:** Para un algoritmo sólido es recomendable emplear técnicas de normalización y ajuste de parámetros para adaptarse a diferentes condiciones de pista.
- **Resistencia a obstáculos:** Para un algoritmo sólido, es necesario implementar estrategias de corrección de trayectoria y planificación de movimiento mediante código.
- **Adaptabilidad a diferentes superficies:** Para un algoritmo sólido es necesario implementar una calibración inicial y la capacidad de aprendizaje automático para mejorar la adaptabilidad ante tipos de línea no convencionales.
- **Estabilidad en velocidad y curvas:** Para un algoritmo sólido, un adecuado tuning del control PID permitirá controlar la velocidad y el ajuste de dirección en la navegación sin oscilaciones que afecten su movilidad.

[26]

H. Algoritmos implementados para el seguimiento de líneas

Existen algoritmos previamente desarrollados especializados en la detección y seguimiento de líneas, algunos de ellos son los que se mencionan a continuación:

- **SIFT (Scale-Invariant Feature Transform):** Detecta y describe características invariantes a la escala en una imagen, especializado en el seguimiento en movimiento y en la caracterización de puntos clave en el objeto.
- **SURF (Speeded-Up Robust Features):** La base del algoritmo es similar a SIFT con la diferencia de ser más rápido, lo que le ha permitido estar presente en diferentes aplicaciones de seguimiento visual.
- **ORB (Oriented FAST and Rotated BRIEF):** Combina la detección rápida de características (FAST) con la descripción de características BRIEF. Es eficiente y adecuado para el seguimiento de objetos en tiempo real.
- **PID (Control Proporcional Integrador Derivativo):** Aunque propiamente no es un algoritmo especializado en este rubro, es necesario para controlar el movimiento, permitiendo el ajuste de la dirección y velocidad del robot.
- **CNN (Convolutional Neural Networks):** Una aplicación más avanzada y robusta implica el uso de Redes Neuronales Convolucionales, las cuales aprenden a detectar líneas y seguir las, sin embargo, requieren de un conjunto grande de imágenes y de mucho tiempo de entrenamiento.

[27]

VIII. UNIDADES FUNCIONALES - APRENDIZAJE PROFUNDO

A. Redes Neuronales Convolucionales (CNN)

Las Redes Neuronales Convolucionales son una clase de redes neuronales profundas ampliamente utilizadas para el análisis de imágenes. Las CNNs se componen por una o más capas conectadas como en la red neuronal estándar. Este tipo de Red está diseñada para aprovechar las propiedades 2D de las imágenes de entrada, las cuales tienden a tener problemas las redes convencionales.

Las capas convolucionales de las redes aplican un conjunto de filtros a la imagen de entrada, cada uno de ellos es pequeño en cuanto al ancho y alto, pero extendido en la profundidad completa de la imagen de entrada. Durante el paso hacia adelante, cada filtro se desliza a través del ancho y alto de la imagen de entrada para producir un mapa de características 2D.

Tras la capa de convolucional, la CNN aplica una operación llamada "ReLU" (Rectified Linear Unit) introduce no linealidades en el modelo, con el fin de permitirle a la red aprender de representaciones más complejas. Después de esto, se aplica una operación de "Pooling" o agrupación enfocada en reducir la dimensionalidad especial (ancho y alto) de la imagen de entrada, el objetivo de esta operación es disminuir la cantidad de parámetros para prevenir el sobre ajuste.

Tras varias capas convolucionales y de pooling, la imagen de entrada se transforma en una matriz de píxeles en bruto a una representación que la red pueda interpretar, las capas conectadas al final de la red son empleadas para clasificar la imagen en las clases definidas a partir de la representación interpretada.

1) *Código de ejemplo de una CNN en Python:* A continuación representamos una manera de estructurar una Red Convolutacional en Python a través del uso de la librería de Keras:

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Inicializar la CNN
modelo = Sequential()

# Primera capa de convolución y pooling
modelo.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3),
activation = 'relu'))
modelo.add(MaxPooling2D(pool_size = (2, 2)))

# Segunda capa de convolución y pooling
modelo.add(Conv2D(32, (3, 3), activation = 'relu'))
modelo.add(MaxPooling2D(pool_size = (2, 2)))

# Aplanar
modelo.add(Flatten())

# Capa completamente conectada
modelo.add(Dense(units = 128, activation = 'relu'))
modelo.add(Dense(units = 1, activation = 'sigmoid'))

# Compilar la CNN
modelo.compile(optimizer = 'adam',
loss = 'binary_crossentropy', metrics = ['accuracy'])
```

[32]

2) *Entrenamiento de red YOLO (You Only Look Once):* YOLO (You Only Look Once) es una serie de modelos de detección de objetos en tiempo real desarrollados por Ultralytics.

Estos modelos están diseñados para ser rápidos y precisos, permitiendo identificar y localizar objetos en imágenes y videos de manera eficiente [32].

YOLOv8 (You Only Look Once, versión 8) es una versión mejorada de la serie de modelos YOLO para la detección de objetos en tiempo real. Los modelos YOLO están diseñados para ser rápidos y precisos, permitiendo identificar y localizar objetos en imágenes y videos de manera eficiente. YOLOv8 incorpora varias mejoras y optimizaciones respecto a sus predecesores [33]. Algunas características clave de YOLOv8 son:

- Arquitectura Optimizada: Diseño mejorado del backbone y el neck del modelo para mejorar la eficiencia y la precisión.
- Mayor Precisión: Reducción de falsas detecciones y aumento de la precisión general.
- Velocidad de Inferencia: Optimizaciones para ejecutar de manera más rápida en diferentes hardwares.
- Transfer Learning: Soporte para transfer learning, permitiendo que los usuarios adapten modelos preentrenados a sus propios conjuntos de datos de manera eficiente.

3) *Roboflow*: Roboflow es una plataforma que ayuda a los desarrolladores y equipos de machine learning a preparar, etiquetar y mejorar conjuntos de datos para entrenar modelos de visión por computadora. Ofrece una serie de herramientas y funcionalidades que facilitan el proceso de gestión de datos para proyectos de aprendizaje automático [34], incluyendo:

- Anotación y etiquetado de imágenes.
- Gestión de conjuntos de datos.
- Preprocesamiento de imágenes.
- Implementación y despliegue.
- Colaboración en proyectos.

A continuación, se muestra el código requerido para el entrenamiento de YOLO nano, utilizando RoboFlow para la creación de la base de datos para el entrenamiento.

```
import os
HOME = os.getcwd()

!pip install ultralytics==8.0.196

from IPython import display
display.clear_output()

import ultralytics
ultralytics.checks()

from ultralytics import YOLO

from IPython.display import display, Image

!mkdir {HOME}/datasets
%cd {HOME}/datasets

!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="#")
project = rf.workspace("deteccion-objetos")
.project("deteccionseniales2.0")
version = project.version(1)
dataset = version.download("yolov8")

%cd {HOME}

!yolo task=detect mode=train model=yolov8s.pt
data={dataset.location}/data.yaml epochs=40 imgsz=340
```

```
plots=True cache=True
```

IX. SOLUCIÓN DEL PROBLEMA

Para dar solución a este reto se decidió por hacer la división entre nodos que corren dentro de nuestra computadora, incluyendo nodos como sign_detection y vertical_detection esto con el propósito de reducir el procesamiento que se lleva a cabo dentro de la Jetson Nano, reduciendo el 'overhead' y por ende la latencia que presenta el robot y en consecuencia mejorar el comportamiento del mismo. De igual manera tenemos aquellos nodos que corren dentro de la misma Jetson Nano, entre los que se encuentran los nodos controller, error_line y sign_information. Estos nodos son los encargados de realizar los ajustes de velocidad lineal y angular acorde a las señales detectadas, medir el error respecto a la línea central de la pista con ayuda de la pendiente y hacer el procesamiento de imagen para ser enviado a los nodos de inferencia o 'detection' respectivamente.

1) Nodos Laptop

:

Sign Detection

En este nodo se hace uso del modelo preentrenado en YOLO V8 nano de 340x340 píxeles con un conjunto de datos de 8 mil imágenes con un total de 12 clases, las señales a detectar así como la línea punteada o 'dot line' del cruce, con el objetivo de poder identificar las condiciones de cruce, brindando mayor robustez a nuestra implementación.

El código comienza cargando el modelo preentrenado dentro de la variable model.

```
self.model = YOLO('.../DeteccionSeniales4_340.pt')
```

Posteriormente, se realiza la creación de los suscribers, tanto a /sign_information el cual es el encargado de realizar la lectura de la imagen pre-procesada con una frecuencia de 10 Hz, así como el suscriber a /vertical_bool para la detección de las líneas verticales de cruce, fragmentos de este código se muestran a continuación:

```
# Realiza la suscripción de la imagen
self.subscription = self.create_subscription(
    Image,
    '/sign_information',
    self.camera_callback,
    10)

# Realiza la suscripción a la línea vertical
self.subscription = self.create_subscription(
    Dotline,
    '/vertical_bool',
    self.vertical_callback,
    10)
```

Posteriormente, se realiza la creación de los publicadores, el primero para la inferencia del modelo YOLO V8, el segundo un arreglo de las señales detectadas, indicando aquella más cercana, así como la publicación de la imagen con los resultados de la inferencia, cuyo principal propósito es poder observar el comportamiento y precisión del modelo. Fragmentos de este código se muestran a continuación:

```
self.yolov8_pub = self.create_publisher(Yolov8Inference,
"/Yolov8_Inference", 1)
self.pred_pub = self.create_publisher(Signal, "/signal_bool", 1)
self.img_pub = self.create_publisher(Image, "/inference_result", 1)
```

De igual manera se crea un timer timer_callback_signs que es usado para realizar la inferencia del modelo.

```
self.timer = self.create_timer(0.2, self.timer_callback_signs)
```

De igual manera se realiza la declaración y definición de las banderas de detección de la señal de 'stop', 'dotLine', 'lineLeft', 'lineRight' y 'lineBoth'.

Camera callback Convierte el mensaje de imagen ROS a una imagen OpenCV si la información en data no es nula.

Vertical callback Recibe la información de la línea vertical detectada, tanto del lado izquierdo y derecho de la imagen, así como si se detecta en ambos y posteriormente los guarda en las variables booleanas previamente declaradas.

Timer callback signs En esta función se obtiene el nombre de las clases detectadas en la inferencia de nuestro modelo con base en la imagen recibida desde la cámara del PuzzleBot, así como las coordenadas de su 'bounding box' para ser adjuntadas dentro de `yolov8_inference`, repitiendo este procedimiento para cada uno de los resultados obtenidos en la detección. Fragmentos de este código se muestran a continuación:

```
def timer_callback_signs(self):
    results = self.model(source=self.img, conf=0.4,
    verbose=False)
    for r in results:
        boxes = r.boxes
        for box in boxes:
            self.inference_result = InferenceResult()
            b = box.xyxy[0].to('cpu').detach().numpy().copy()
            c = box.cls
            self.inference_result.class_name = self.model.names[int(c)]
            self.inference_result.top = int(b[1])
            self.inference_result.left = int(b[0])
            self.inference_result.bottom = int(b[3])
            self.inference_result.right = int(b[2])
            self.yolov8_inference.yolov8_inference.append(self.inference_result)

    self.escribirMensaje(self.yolov8_inference)
    self.pred_pub.publish(self.senialesDetectadas)
```

Escribir mensaje Esta es la función principal de `sign_detection`, ya que se encarga de determinar el estado de las banderas de detección de señales. Para esto se hace un recorrido por cada una de las señales detectadas por el modelo. Para la detección de 'dotLine', se hace una discriminación, si el modelo detecta estar sobre la intersección gracias a la detección de 'dotLine' en el modelo, se mantiene activa esta bandera por un periodo de 3 segundos, tras los cuales debe pasar 15 segundos para reiniciar su condición de detección. Fragmentos del código se muestran a continuación:

```
for inference in yoloInference.yolov8_inference:
    if class_name == "dotLine":
        if nearest > 200: #Distancia del robot
            self.senialesDetectadas.dot_line = True
        else:
            if elapsed_time < 3.0:
                self.senialesDetectadas.dot_line = True
            elif elapsed_time >= 15.0:
                self.dot_line_sent = False
                self.dot_line_detected_time = None
```

De igual forma, de detectar el semáforo en un rango mayor a 120 píxeles (cercanía del robot a la señal) las banderas correspondientes se activan.

```
if class_name == "greenLight":
    if nearest > 120:
        self.senialesDetectadas.green_light = True
elif class_name == "redLight":
    if nearest > 120:
        self.senialesDetectadas.red_light = True
elif class_name == "yellowLight":
    if nearest > 120:
        self.senialesDetectadas.yellow_light = True
```

Posteriormente, se compara la señal detectada con mayor área diferente de 'dotLine', para que dependiendo de la señal detectada se cumplan una de tres condiciones, primero que únicamente se active la bandera de la señal correspondiente,

como 'roadwork' o 'roundabout' con un condicional de distancia, segunda condición, como es el caso de la señal 'stop' y 'giveWay' se inicia un contador de tiempo transcurrido para determinar cuándo activar o desactivar dicha bandera y por último que requieran de la detección de una señal previa para poder activarse, como es el caso de 'lineLeft', 'lineRight' y 'lineBoth' que requieren de la previa activación de roundabout. Ejemplos de activación directa:

```
if class_name == "aheadOnly":
    self.senialesDetectadas.ahead_only = True
elif class_name == "roadwork":
    if signal_with_max_area.bottom > 120:
        self.senialesDetectadas.roadwork = True
elif class_name == "roundabout":
    self.senialesDetectadas.roundabout = True
```

Ejemplos de activación de acuerdo con el tiempo transcurrido:

```
elif class_name == "stop":
    if signal_with_max_area.bottom > 110: #cercanía
        # Si detectamos "stop", verificamos el tiempo
        if not self.stop_signal_sent:
            self.stop_detected_time = time.time()
            self.senialesDetectadas.stop = True
            self.stop_signal_sent = True
        else:
            elapsed_time = time.time() - self.stop_detected_time
            if elapsed_time < 5.0:
                self.senialesDetectadas.stop = True
            elif elapsed_time >= 12.0:
                self.stop_signal_sent = False
                self.stop_detected_time = None
elif class_name == "giveWay":
    # Se mantiene en dotline por tres segundos
    if self.dot_line_detected_time:
        elapsed_time = time.time() - self.dot_line_detected_time
        if elapsed_time > 3.0:
            self.senialesDetectadas.give_way = True
            self.senialesDetectadas.dot_line = False
```

Ejemplos de activación con condicional previa:

```
if self.senialesDetectadas.roundabout:
    if self.lineaLeft:
        self.senialesDetectadas.turn_left = True
    elif self.lineaRight:
        self.senialesDetectadas.turn_right = True
    elif self.lineaBoth:
        self.senialesDetectadas.turn_right = True
```

Por último, se realiza la publicación de las señales detectadas.

```
self.pred_pub.publish(self.senialesDetectadas)
```

Vertical Detection

De igual manera que con Sign Detection, se realiza la carga del modelo de inferencia para la detección de las líneas verticales de cruce.

```
self.model = YOLO('.../dotVertical.pt')
```

Esto con el objetivo de poder determinar la dirección de giro ante la presencia de la señal 'roundabout' ante cualquiera de las direcciones en las que se aproxime el robot al cruce, brindando mayor robustez a la implementación diseñada.

Posteriormente, se hace la creación de los publishers correspondientes, de una manera muy similar a como se realizó en Sign Detection como se puede observar en IX-1, únicamente añadiendo la condición vertical. De igual forma que en el nodo Sign Detection se realiza la función `camera_callback` como se puede observar en IX-1, así como el `timer_callback_signs` como se observa en IX-1.

La principal diferencia se encuentra en la función '**Escribir mensaje**', ya que en este caso haciendo un recorrido por cada inferencia del modelo, se calcula el centroide de las detecciones, comparando este valor con el centro de la imagen, y en caso de ser mayor se manda 'lineaRight', de lo contrario se manda 'lineaLeft'.

```
def escribirMensaje(self, yoloInference):
    for inference in yoloInference.yolov8_inference:
        if class_name == "Vertical-dotline":
            if center_x > self.img_center:
                self.senialesDetectadas.right = True
            else:
                self.senialesDetectadas.left = True
```

Y en caso de detectarse ambos, se desactivan ambas banderas y se manda 'lineaBoth'.

```
def escribirMensaje(self, yoloInference):
    for inference in yoloInference.yolov8_inference:
        if self.senialesDetectadas.left and
           self.senialesDetectadas.right:
            self.senialesDetectadas.left = False
            self.senialesDetectadas.right = False
            self.senialesDetectadas.both = True
```

Por último se publica el resultado.

```
self.pred_pub.publish(self.senialesDetectadas)
```

2) Nodos Jetson Nano

Controller

Este nodo es el encargado de recibir los valores booleanos de las banderas, así como el error de línea, esto con el fin de hacer los ajustes correspondientes a la velocidad lineal, así como a la velocidad angular con ayuda de un controlador PI. Empezamos creando un nodo publicador de velocidades con ayuda de los mensajes Twist.

```
self.pred_pub.publish(self.senialesDetectadas)
```

De igual manera se creó un timer llamado `timer_callback_controller`, siendo este timer el encargado de llamar a la función correspondiente para la asignación de las diferentes velocidades. También se realizan las suscripciones adecuadas, tanto al tópico `error_line` así como a `signal_bool`.

```
self.subscription_light = self.create_subscription(
    Float32,
    'error_line',
    self.line_error_callback,
    rclpy.qos.qos_profile_sensor_data )

# Suscripción que lee las señales detectadas por YOLO
self.subscription_seniales_detectadas =
self.create_subscription(
    Signal,
    '/signal_bool',
    self.deteccion_callback,
    rclpy.qos.qos_profile_sensor_data )
```

Line error callback En esta función se reciben los datos de 'error line' y se asigna a una variable 'errorLinea', de igual manera realiza los ajustes a la bandera de 'lecturaError' para determinar que se ha podido recibir un dato.

```
def line_error_callback(self, msg):
    #Si el dato es nulo no se guarda
    if msg is not None:
        self.errorLinea = msg.data
        self.lecturaError = True
```

Deteccion callback De igual manera, esta función recibe los valores de '/signal_bool' y en caso de no ser nulos, son guardados en la variable 'senialesBool'

```
def deteccion_callback(self, msg):
    #Si el dato es nulo no se guarda
    if msg is not None:
        self.senialesBool = msg
```

Timer callback controller Esta es la principal función del nodo, la cual es llamada por el timer creado con anterioridad. Es la función encargada de realizar los ajustes de velocidad con base en los datos recibidos de la cámara, tanto el error a la línea, así como las señales detectadas.

Se inicia comparando si se detecta 'dotLine' y la bandera 'cruce' es falsa, ajustando las banderas 'cruce' a verdadero y 'senialCruce' a falso.

```
if self.senialesBool.dot_line and not self.cruce:
    self.cruce = True
    self.senialCruce = False
```

De ser el caso de que la bandera de cruce sea verdadera, se entra en un ciclo en el cual se toma el tiempo actual con ayuda de la función 'time()', de igual manera se compara que el valor de 'senialCruce' sea falsa, de serlo se entra en un ciclo en el que se comparan las señales detectadas, cambiando los valores de las variables 'tiempo deseado recta' y 'tiempo deseado angular' los cuales se obtuvieron tomando en cuenta el tiempo que le toma al robot realizar cada uno de los giros necesarios correspondientes a la señal detectada. Ejemplos de este código se muestra a continuación:

```
def timer_callback_controller(self):
    if self.cruce and not self.senialCruce:
        self.tiempo_actual = time.time()
        if self.senialesBool.ahead_only:
            self.tiempo_deseado_recta = 3.5
            self.tiempo_deseadoAngular = 0.0
            self.tiempo_deteccion = self.tiempo_actual
            self.senialCruce = True
    ....
```

En caso de que la variable 'senialCruce' sea verdadera, se entra en un ciclo en el que se compara que si el tiempo deseado de recta más el tiempo de detección es mayor al tiempo actual, se establece una velocidad lineal de 0.1 y una velocidad angular de cero.

```
if self.tiempo_deseado_recta+self.tiempo_deteccion
> self.tiempo_actual:
    self.velL = 0.1
    self.velA = 0.0
```

de lo contrario, se calcula la variable tiempo necesario como la suma del tiempo deseado de recta más el tiempo deseado angular más el tiempo de detección, la lógica siendo que se guarda el tiempo total requerido para el tiempo de recta, así como el tiempo angular más el tiempo de detección como el tiempo total necesario que el robot requiere mantener una velocidad angular para dar el giro correspondiente, un ejemplo del código se muestra a continuación:

```
else:
    tiempoNecesario = self.tiempo_deseado_recta
    +abs(self.tiempo_deseadoAngular)+self.tiempo_deteccion

    if self.tiempo_deseadoAngular > 0:
        if tiempoNecesario > self.tiempo_actual:
            self.velL = 0.0
            self.velA = 0.1
        else:
            self.cruce = False
            self.senialCruce = False
```

En este caso se toma el tiempo deseado angular como un absoluto, ya que utilizamos el signo para determinar la dirección de rotación.

Una vez terminando estas comprobaciones para los casos de cruce, se toman las siguientes condiciones para el seguimiento de línea. En el caso de tener un error en un rango de más menos 0.05 se mantiene una velocidad lineal constante y una velocidad angular de 0, esto con el objetivo de que el Puzzlebot no esté oscilando siempre y tenga cierto margen para el error.

```
#Condición de que debe seguir la línea
else:
```

```

# Se hace una aceptación del error para que no oscile
if self.errorLinea >= -0.05 and self.errorLinea <= 0.05
    self.velA = 0.0

De lo contrario, si el error es mayor a 0.05, se realiza
un control proporcional y derivativo (PD) con una constante
proporcional de 0.2 y una constante derivativa de 0.016.
Acotando la velocidad angular a un valor máximo de más
menos 0.2.

else:
    #Se omiten cálculos de 'proportional' y 'derivative'
    # Calcular la señal de control
    self.velA = proportional + derivative

    #Se acota la velocidad positiva
    if self.velA > 0.2:
        self.velA = 0.2

    #Se acota la velocidad negativa
    if self.velA < -0.2:
        self.velA = -0.2

    De igual forma establece una condición en la que no se
recibe ningún error, el robot se detiene, utilziado principalmente
mientras se inicializa el nodo de seguidor de línea.

    # No avanza hasta que lea del tópico error
    if self.lecturaError:
        self.velL = 0.08

    else:
        self.velL = 0.00
        self.velA = 0.00

```

Por último, se establecen las velocidades tanto para las señales 'stop', 'red light', 'roadwork' y 'yellow light', con una velocidad lineal de cero para las primeras dos y una velocidad de 0.04 (mitad de la velocidad habitual) para las últimas dos.

```

#Condiciones de detección de señales generales
if self.senialesBool.roadwork:
    self.velL = 0.04
elif self.senialesBool.yellow_light:
    self.velL = 0.04
elif self.senialesBool.stop:
    self.velA = 0.0
    self.velL = 0.0
elif self.senialesBool.red_light:
    self.velA = 0.0
    self.velL = 0.0
    self.senialCruce = False

```

Para posteriormente enviar dichas velocidades por el mensaje Twist() a través del publicador creado anteriormente.

```

# Crear el mensaje Twist y publicarlo
twist_msg = Twist()
twist_msg.linear.x = self.velL
twist_msg.angular.z = self.velA
self.pub_cmd_vel.publish(twist_msg)

```

Error line

Para este nodo se empieza creando la suscripción a la cámara con ayuda del tópico '/video_source/raw' así como la creación de dos publicadores, el primero de los cuales se encargará de enviar el error calculado con ayuda del tópico 'error_line' y el segundo encargándose de publicar la imagen con los rectángulos correspondientes a la línea y el error con propósito de debugging.

```

self.sub_img = self.create_subscription(Image,
'/video_source/raw', self.image_callback,
rclpy.qos.qos_profile_sensor_data)
self.pub_error = self.create_publisher(Float32,
'error_line', 10)
self.pub_line_image = self.create_publisher(Image,
'/_img_line', 10)

```

Posteriormente, se crea el timer encargado de llamar a la función line_detection_callback la cual tomará la

imagen y realizará el procesamiento de imágenes correspondientes para la obtención del error.

```

self.timer = self.create_timer(self.timer_period,
self.line_detection_callback)

```

Image callback Toma la imagen del tópico correspondiente y lo convierte del formato de imagen de ROS a CV2 y lo guarda en la variable 'camaraImg' y cambia la bandera de lectura de imagen a verdadero.

```

def image_callback(self, msg):
    if msg is not None:
        self.camaraImg = self.bridge.imgmsg_to_cv2(msg, "bgr8")
        self.imgLecture = True
    else:
        self.imgLecture = False

```

Line detection callback Esta función llama a la función calculoError para su posterior publicación por el tópico 'error line' a través de un mensaje personalizado 'errorMsg', de lo contrario lanza un mensaje indicando que hubo un fallo en el procesamiento de la imagen.

```

def line_detection_callback(self):
    if self.imgLecture:
        self.errorMsg.data = self.calculoError(self.camaraImg)
        self.pub_error.publish(self.errorMsg)
    else:
        self.get_logger().info(f'Failed to process image')

```

calculoError En esta función se llama a la función resize_image recibiendo como argumento la imagen recibida, así como a la función preprocess recibiendo como argumento la imagen redimensionada y recortada y por último llama a la función pendiente_centroides que recibe la imagen binarizada y la imagen preprocesada. Retornando al final el error.

```

def calculoError(self, img):
    self.resize_image(img)
    self.preprocess(self.imagenCortada)
    error = self.pendiente_centroides(self.imagenBN,
    self.imagenCortada)
    return error

```

Resize image Esta función toma la imagen raw recibida de la cámara y la redimensiona a un medio del valor original para posteriormente recortar la mitad de la misma, quedándonos con la parte de la imagen más cercana al robot, por último recorta el primer y último octavo de la imagen, esto con tal de evitar errores de detección debido a las líneas de los extremos de la pista.

```

def resize_image(self, img):
    ancho_r = img.shape[1] // 2
    alto_r = img.shape[0] // 2
    img_redimensionada = cv2.resize(img, (ancho_r, alto_r))

    #Se recorta la imagen
    alto_original = img_redimensionada.shape[0]
    ancho_original = img_redimensionada.shape[1]
    inicio_y = 0 # la mitad del alto para el inicio del corte
    fin_y = alto_original-int(alto_original/2)
    inicio_x = int(ancho_original // 8)
    fin_x = int(7 * ancho_original // 8)
    imgC = img_redimensionada[inicio_y:fin_y, inicio_x:fin_x]
    self.imagenCortada = imgC

```

Preprocess Esta función aplica un filtro de media a la imagen para posteriormente ser convertida a escala de grises, es así que se realiza la binarización con el método de Otsu así como un 'bitwise not' para invertir la imagen. Retornando la imagen binarizada.

```

# Función que hará el preprocesamiento
def preprocess(self, imgC):
    # Se calcula el filtro medio
    filtro_medián = cv2.medianBlur(imgC, 3)
    #Imagen a escala de grises
    img_g = cv2.cvtColor(filtro_medián, cv2.COLOR_BGR2GRAY)
    # Binarización de tipo Otsu
    _, imagen_binarizada = cv2.threshold(img_g, 0, 255,
    cv2.THRESH_BINARY+cv2.THRESH_OTSU)

```

```

imagen_binarizada = cv2.bitwise_not(imagen_binarizada)
self.imagenProcesada = imagen_binarizada
self.imagenBN = imagen_binarizada

```

Pendiente centroides Esta función procesa una imagen binaria aplicando una operación de erosión para reducir el ruido. Luego, encuentra todos los contornos en la imagen erosionada y los evalúa. Si hay múltiples contornos, determina el vértice más bajo de cada contorno y selecciona aquellos cercanos a la cámara. De estos candidatos, selecciona el contorno con el área más grande. Utiliza el rectángulo mínimo que encierra este contorno para calcular la posición del centro en el eje x y determina un error basado en la diferencia entre esta posición y el punto central de la imagen, normalizando este valor por el ancho de la imagen. Finalmente, dibuja el rectángulo y el valor del error en la imagen original, almacena la imagen procesada y retorna el error calculado. Fragmentos de este código se muestran a continuación:

```

def pendiente_centroides(self, img_bn,image):
    contours_blk, _ =
        cv2.findContours(morf_d3.copy(),cv2.RETR_TREE,
        cv2.CHAIN_APPROX_SIMPLE)
    if num_contours > 0:
        for con_num in range(num_contours):
            blackbox = cv2.minAreaRect(contours_blk[con_num])
            box = cv2.boxPoints(blackbox)
            min_y = 120
            for i, (x_box, y_box) in enumerate(box):
                if y_box <= min_y:
                    min_y = y_box
                if min_y < 20:
                    candidates.append((min_y,con_num))
            candidates = sorted(candidates)
        for neary, contornNear in candidates:
            contour = contours_blk[contornNear]
            area = cv2.contourArea(contour)
            if area > max_area:
                max_area = area
                max_contour = contour
    return error

```

Sign information

Este nodo comienza creando un suscriber que recibe la imagen de la cámara mediante el tópico /video_source/raw a su vez de crear un publisher a través del tópico /sign_information, así como un timer que llama a la función timer_callback_signs.

```

self.subscription = self.create_subscription(Image,
'/video_source/raw',self.camera_callback,10)
self.img_pub = self.create_publisher(Image,
"/sign_information", 10)
self.timer = self.create_timer(self.timer_period,
self.timer_callback_signs)

```

Camera callback Esta función recibe la imagen de la cámara y la convierte a formato CV2, de igual manera realiza una rotación de 180° y un redimensionamiento de 340 por 340 píxeles.

```

if data is not None:
    self.img = bridge.imgmsg_to_cv2(data, "bgr8")
    self.img = cv2.rotate(self.img, cv2.ROTATE_180)
    self.img = cv2.resize(self.img, (340, 340))

```

Timer callback signs Esta función toma la imagen procesada y realiza la conversión de CV2 a ROS y la publica por el nodo /sign_information

```

def timer_callback_signs(self):
    img_msg = bridge.cv2_to_imgmsg(self.img, encoding="bgr8")
    self.img_pub.publish(img_msg)

```

Mensajes personalizados

Para este proyecto se crearon dos mensajes personalizados, el primero 'Signal.msg' como un arreglo de 11 variables booleanas, cada una representando diferentes señales de tráfico como valores verdadero y falso.

```

bool ahead_only
bool turn_right
bool turn_left
bool roundabout
bool give_way
bool roadwork
bool stop
bool red_light
bool yellow_light
bool green_light
bool dot_line

```

De igual manera se creó el mensaje 'Dorline.msg' que contiene un arreglo de tres variables booleanas representando la posición de detección de las 'vertical dot lines' para las intersecciones.

```

bool both
bool right
bool left

```

X. RESULTADOS

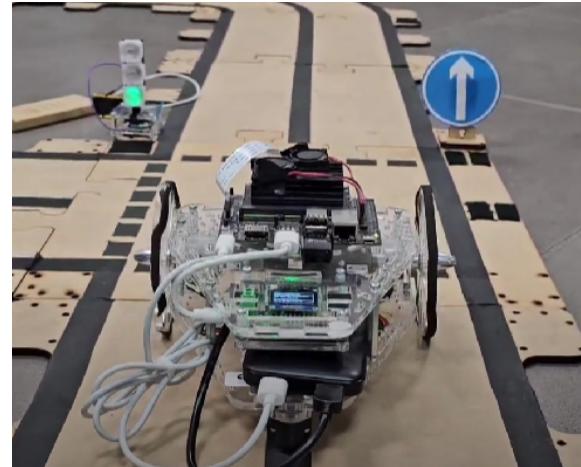


Fig. 1. Funcionamiento Robot

A. Interpretación de resultados

Al implementar nuestra solución, obtenemos básicamente un vehículo autónomo capaz de desplazarse a través del seguimiento de una línea, además de que simultáneamente detecta señales y actúa dependiendo de la señal. En el video mostrado en la figura 1, relativo al funcionamiento del puzzlebot, se puede observar cómo se ejecuta el seguidor de línea con la función de contornos, detectando la línea con mayor área para hacer el seguimiento de esa línea. Asimismo, se muestra cómo se calcula un error dependiendo de la pendiente, ya sea positivo o negativo, para activar una velocidad angular positiva o negativa, mientras que mantiene una velocidad lineal constante. Con respecto a la detección de señales, entrenamos la última capa de nuestro modelo utilizando yolov8n.pt, tomando alrededor de 3 mil imágenes, etiquetándolas con su señal correspondiente o como fondo. En cuanto al controlador,

con el seguidor de línea solo enviamos al controlador el error obtenido de la línea respecto al centro de la imagen. Este error ya está normalizado y se le añade el control con un "PD" (Proporcional y Derivativo). De esta forma, solo modificamos la velocidad angular con el error y definimos límites para evitar que el microcontrolador se reiniciara. En relación con la detección de señales, enviamos al controlador solo un valor de tipo booleano para indicar que se está detectando dicha señal y ejecutar una acción específica. Además, en el controlador, cuando se está en un cruce, utilizamos el tiempo en lugar de la odometría para avanzar o girar según la señal, con el objetivo de reducir el procesamiento, enviar menos información a través de los tópicos y usar menos nodos. De este modo, ajustamos el tiempo necesario para cumplir con la trayectoria, ya sea para seguir recto o para llegar al centro del cuadro y realizar un giro, todo lo cual se puede apreciar en el video 1. Asimismo, para continuar robusteciendo nuestro modelo, detectamos la línea punteada vertical para determinar si el PuzzleBot podía girar en esa dirección. En caso de que la señal estuviera incorrecta o la detección no fuera la adecuada, rectificábamos si se podía realizar algún giro hacia la dirección solicitada. Finalmente, en el video se puede observar el recorrido completo y cómo se aplica todo lo mencionado anteriormente.

B. Robustez de implementación

Roundabout Izquierda

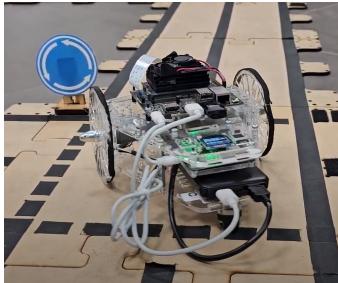


Fig. 2. Roundabout Izquierda

En este caso, el robot es capaz de detectar que no existe opción de giro a la derecha, ya que únicamente se detecta una 'vertical dot line' del lado izquierdo, por lo que gira en dicha dirección. Figura [2].

Roundabout Derecha

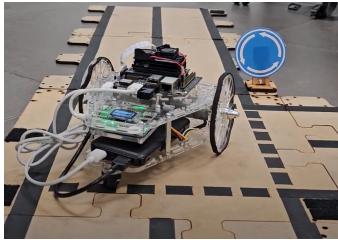


Fig. 3. Roundabout Derecha

En este caso, el robot es capaz de detectar que la única dirección de giro permitida para completar otra vuelta es con

un giro a la derecha, esto gracias a la detección de una única 'vertical dot line' en ese sentido. Figura [3].

Give Way

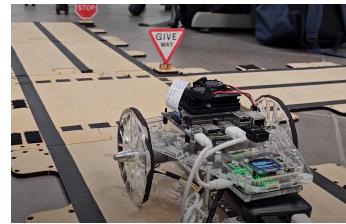


Fig. 4. Give Way

En este caso el robot se detiene al detectar la señal de 'Give Way' dentro de la intersección por un periodo de tres segundos, simulando dar el paso, tiempo tras el cual sigue su camino. Figura [4].

Right Signal Edge Case

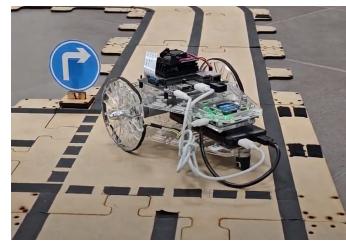


Fig. 5. Right Signal Edge Case

En este caso el robot detecta la señal 'turnRight' pero dado que únicamente se detecta la 'vertical dot line' del lado izquierdo, este realiza el giro en esta dirección, descartando la señal, esto con el objetivo de que este no se salga de la pista. Figura [5].

XI. CONCLUSIONES

El producto final de esta actividad y, por consiguiente, de este bloque, Implementación de robótica inteligente representa la unión de múltiples elementos desarrollados a lo largo de estas semanas de manera individual con propósitos específicos, los cuales con más o con menos áreas de mejora en cuanto a la manera de solventar las necesidades de su implementación y entorno, semana a semana nos encaminaban a una labor final: interactuar con el entorno de manera autónoma, un desafío increíblemente mayor no solo por el reto en sí mismo que implica el desarrollar una red lo suficientemente capaz de identificar los elementos a su alrededor, asociarlos a una clase con precisión para después ser retomados por el controlador del robot para ajustar su cinemática con base en una lógica asociada a cada clase, sino también computacionalmente hablando.

Las limitantes lejos de frenar el desarrollo del producto final nos permitieron dar nuevos enfoques de solución aprovechando esos vacíos en cuanto a poder de cómputo para centrarlos en tareas concretas, dividiendo el procesamiento de imágenes para la identificación de señales como una tarea de

la cámara del PuzzleBot apoyada por el poder de cómputo de nuestras computadoras mientras que los cálculos cinemáticos y de lógica tras identificada la línea, curvas o señales lo hacía la Jetson Nano. Aunado a esto, creamos una lógica de interacción ante las posibles señales y recorridos posibles a desarrollar en una pista, que aunque limitados a una cierta cantidad visible para nosotros, no lo era tan sencillo para el robot, teniendo que usar los escasos elementos (particularmente el de la intersección y la línea punteada) para actuar ante la situación que se le presentara al robot.

Quizás lo que llevó más tiempo significativamente hablando fue el entrenamiento de nuestro modelo a través de Roboflow, especializado en el etiquetado y construcción del dataset de señales, líneas y semáforos, y YOLO, especializada para el entrenamiento de la misma y la construcción de nuestro modelo de CNN, estando a lo largo de estas 3 semanas de desarrollo desde el primer modelo funcional hasta el último con el fin de solventar fallas de los anteriores o introduciendo nuevas clases y variables que poco a poco en el desarrollo de los nodos restantes fueron saliendo. Las mejoras y correcciones con respecto a los productos particulares pasados se dieron en el seguidor de línea (siguiendo la misma lógica del anterior pero actuando con mayor precisión dada la nueva altura de la cámara a costa de oscilar un poco más y la toma de las curvas) y el PID ajustado a la lógica de nuestro proyecto, la cual fue desarrollada tras consolidar un modelo de detección de señales lo suficientemente robusto como para probarlo en un entorno más cercano a la situación final a solventar. Para ser honestos, este proyecto requirió dar más tiempo del que anteriores retos habíamos destinado, sin embargo, dada la situación, fue entendible dar todo lo necesario para llegar al resultado final.

Cada reto anterior completado con su respectivo grado de dificultad, aunque funcional, siempre hubo áreas de mejora que nos llevaron a productos más robustos a posteriori, elementos que se ven reflejados en todos los aspectos donde nuestra visión computacional falló en los casos anteriores, y aunque este solventa con creces esas carencias, aún vemos áreas de oportunidad en este producto final, elementos que a una mayor cantidad de tiempo de desarrollo hubiéramos podido llegar a soluciones todavía más robustas, pero con el mismo ingenio que pudimos usar para darle solución en esta ocasión a estas dudas de comportamiento, sobre todo vemos una área importante de oportunidad de optimizar el funcionamiento del robot, pues no fue solo una sino varias veces en donde este se comportaba contrario a lo esperado aún con el código correcto cargado y ejecutándose en él, así mismo la señales de Turn Right y Turn Left fueron en muchos casos un elemento de incertidumbre de comportamiento, no tanto en la lógica sino más bien en como el modelo podía a veces confundirlas por sus contrarias al encontrarlas durante el recorrido de la pista.

Pese a estas dificultades, el exhaustivo y constante trabajo detrás que en estas últimas tres semanas llevamos a cabo el día de hoy nos permiten entregar un producto lo suficientemente sólido y del que nos sentimos orgullosos de presentar, viendo en él un claro progreso desde el día cero de desarrollo

hasta el día de entrega del mismo donde es evidente como cada una de las cosas aprendidas para visión computacional, procesamiento de imagen, odometría y control convergen en un elemento móvil que interactúa con todo aquello para lo que fue desarrollado para ver y actuar conforme a ello.

XII. REFLEXIONES PERSONALES

1) *Alejandro Gómez*: Haciendo una retrospectiva profunda, considero que la dificultad de cada reto semanal, así como del proyecto final, representó un salto enorme en comparación con los productos entregados el semestre pasado. Sin embargo, agradezco la estructura de los desafíos, ya que nos guiaron semana a semana hacia el desarrollo del producto final. Los retos fueron numerosos y, con cada avance, surgían nuevas problemáticas que nos hicieron pensar y trabajar arduamente para superarlas mediante software y hardware.

A lo largo del proceso, desarrollamos funcionalidades innovadoras que mejoraron el rendimiento del sistema. Implementamos una función de tiempo para el movimiento en las intersecciones y redimensionamos las imágenes dos veces para acelerar el procesamiento. Aunque no todos los intentos fueron exitosos, como el intento de aumentar el módulo de wifi con un nuevo componente que no interactuó como esperábamos, cada esfuerzo nos enseñó algo valioso.

A pesar de los progresos, aún tengo algunas dudas, especialmente en el procesamiento de imágenes, particularmente con la detección de señales de "Turn Right" y "Turn Left", ya que hubo confusiones constantes entre ambas. Esta es un área en la que claramente hay espacio para mejorar, al igual que encontrar componentes que permitan optimizar el funcionamiento del bot, pues aunque logramos reducir el tiempo de respuesta, aún llega a fallar cuando se sobrecalienta, lo cual no solo retrasa el progreso, sino también genera escenarios de error que muchas veces no responden a los cambios implementados en los nodos diseñados.

Este proyecto no solo me ha permitido aplicar conocimientos técnicos, sino también desarrollar habilidades en resolución de problemas y trabajo en equipo. Cada desafío superado fue una oportunidad de aprendizaje y crecimiento, dejándome una profunda satisfacción y entusiasmo por futuros proyectos. La experiencia ha sido invaluable, demostrando que con dedicación y esfuerzo, podemos abordar y superar incluso los retos más complejos. Estoy emocionado por aplicar lo aprendido en futuros desafíos y continuar mejorando mis habilidades.

2) *Abraham Ortiz*: Después de concluir con la situación problemática, me queda claro que el concepto de crear robots realmente autónomos es muchísimo más complejo de lo que esperaba. Aunque imaginaba que era un gran reto lograr que los robots tuvieran comportamientos reactivos al entorno, al abordar este desafío me he dado cuenta de que, a pesar de haberse realizado en un entorno extremadamente controlado y con el apoyo de las materias que ofrecieron una guía, hubo demasiados puntos de mejora necesarios para que el robot fuera verdaderamente autónomo, incluso en el ambiente más controlado imaginable. No había dimensionado el verdadero reto que implica actualmente debido a varias razones. La

primera es que nunca había considerado el aspecto ético desde esta perspectiva. Por ejemplo, estábamos manipulando un robot muy pequeño, con muy poca probabilidad de causar daño grave a una persona. Cuando el robot experimentaba acelerones o giros fuera de control, no se generaban daños. Sin embargo, al pensar en el peligro que podría representar un vehículo que transporta personas y tiene la fuerza para causar lesiones graves o fatales, me queda clara la gran responsabilidad ética en el diseño de estos sistemas. Otra razón es que, en este caso, el entorno estaba controlado, lo que reducía las posibilidades de eventos inesperados y hacía que el diseño de los algoritmos fuera más sencillo. Sin embargo, fue un desafío enorme. No puedo imaginar la cantidad de robustez necesaria para que un robot pueda actuar por sí solo en cualquier situación. Es tan complejo que, a pesar del nivel de tecnología actual, nadie ha sido capaz de crear un robot 100% autónomo. El último motivo que me gustaría agregar es que nunca había considerado cómo ve el sistema. Algo puede ser muy obvio para los humanos, pero para una máquina no lo será. Es necesario adaptar la lógica de programación y utilizar inteligencia artificial para lograr un comportamiento más cercano al humano. Sin embargo, al intentar usar inteligencia artificial, surgen muchos problemas, principalmente porque se requieren muchos datos y un procesamiento intensivo, que a menudo no puede implementarse en todos los dispositivos. En cuanto a los desafíos enfrentados al resolver la situación problemática, siento que fueron numerosos, pero aun así logramos un gran resultado. A continuación, detallo los principales problemas encontrados. El primer problema fue con el seguidor de línea. La primera propuesta resultó ser poco efectiva, por lo que tuvimos que cambiar varias veces el modelo hasta obtener el más adecuado para nuestro caso. En pocas palabras, enfrentamos problemas como que el robot solo giraba bien hacia un lado, tomaba basura del mismo color que la línea como parte de ella, la luz y el piso influían en la detección, y detectaba manchas en la pista como si fueran la línea. Me di cuenta de la importancia de ser específicos con el algoritmo y de encontrar las características más significativas para asegurar que solo detecte la línea correcta. Otro reto fue la latencia. Al utilizar comunicación por wifi para pasar las imágenes, inicialmente tuvimos una latencia de casi 6 segundos. Al redimensionar la imagen, logramos una respuesta casi instantánea. El último problema significativo fue al intentar correr el modelo de YOLO en el propio puzzlebot. Pronto nos dimos cuenta de que no era viable debido al tiempo de procesamiento, por lo que optamos procesar el modelo desde una laptop para enviar mensajes personalizados con simples booleanos al robot, reduciendo el procesamiento en la Jetson y evitando el sobrecalentamiento para un uso más efectivo y prolongado. En conclusión, aunque fue un reto muy complicado, logramos concluir exitosamente y aprender y reforzar muchos conceptos, como tipos de comunicación, procesamiento de imágenes, odometría, redes convolucionales, entre otros.

3) *Alan Flores:* Como reflexión considero que los retos semanales, así como la entrega final de este bloque fueron

un verdadero desafío, ya que requirieron la integración y correcto funcionamiento de múltiples códigos y su respectivo funcionamiento en ROS con la capacidad de interactuar y comunicarse entre ellos mediante los nodos y tópicos, cada uno encargado de una tarea en específico, como lo fueron el seguidor de línea, que requería del procesamiento de imágenes ante variedad de circunstancias y entornos de iluminación, de igual manera los nodos de reconocimiento de señales, así como de aquel encargado de detectar la intersección. Para los cuales se realizó la captura de 3,275 imágenes para las 12 clases correspondientes y su posterior entrenamiento con YOLO v8, para el cual se entrenaron más de 10 redes diferentes para encontrar aquella que mejor se desempeñara. Considero que una de las mayores dificultades dentro del desarrollo de este proyecto fue la lógica necesaria para cada una de las señales, clasificándolas en aquellas de cruce como 'ahead only', 'turn left', 'turn right' y 'roundabout'. Y aquellas que se pueden detectar a lo largo de toda la pista como el 'stops'

4) *Ulises Hernández:* Como reflexión individual de este proyecto, puedo decir que el reto fue bastante complicado, especialmente en la parte de procesamiento de imágenes. Aunque básicamente todo el reto involucró el procesamiento de imágenes para hacer el seguidor de línea y la detección de señales de tránsito, esta tarea presentó varias dificultades. Los resultados del procesamiento de imágenes pueden verse afectados por diversos factores, como el ruido, la iluminación, e incluso el sobrecalentamiento de la Jetson Nano, que comenzaba a fallar en la detección y en ocasiones se apagaba, impidiendo su uso por un tiempo. Estos problemas dificultaron la detección de señales similares, particularmente con las señales de giro a la izquierda y a la derecha. Además de estos retos, probamos diversos métodos para el seguidor de línea y entrenamos varios modelos para la detección de señales. Finalmente, utilizamos los que nos dieron mejores resultados y logramos concluir el proyecto con éxito. Todo esto nos brindó una perspectiva clara sobre los vehículos autónomos y las dificultades asociadas con el procesamiento de imágenes. A pesar de las dificultades, la experiencia fue muy interesante y enriquecedora.

REFERENCES

- [1] PuzzleBot – Manchester Robotics. (s.f.). [link](#).
- [2] La guía para principiantes para configurar un Jetson Nano en JP4.4. (2020, Diciembre 5). ICHI.PRO. [link](#).
- [3] Kangalow. (2023, Enero 12). Wi-Fi hotspot setup – NVIDIA Jetson Developer Kits. JetsonHacks. [link](#).
- [4] teleop_twist_keyboard - ROS Wiki. (s.f.). [link](#).
- [5] What do X,Y, and Z mean in geometry msgs Twist message in ROS. (s.f.). Stack Overflow. [link](#).
- [6] Twist Message Example and /cmd vel - ROS Answers: Open Source QA Forum. (s.f.). [link](#).
- [7] Working with Twist Messages in Python - COM2009 ROS Labs. (s.f.). [link](#).
- [8] Edisonsasig. (2021, Abril 13). La cinemática en la robótica. Roboticoss. [link](#).
- [9] Edisonsasig. (2023, Julio 14). Modelo cinemático y simulación de un robot móvil diferencial. Roboticoss. [link](#).
- [10] Control Automático Educación (2024, Febrero 13). Control Automático Educación. [link](#).
- [11] Tipos de sistemas de control - Brazo robótico y sus clasificaciones. (s.f.). [link](#).

- [12] Castellanos Pérez, M., Medina, A., Álvaro Hernández, S., Lester, S., Maza, A., León, I., y Revisores, O. (s.f.). Instituto Tecnológico de Tuxtla Gutiérrez Reporte Final Residencia Profesional: Desarrollo y Aplicación de Algoritmos de Cooperación en Robots Móviles. link.
- [13] Cortés, U., Castañeda, A., Benítez, A., y Díaz, A. (2015). Control de Movimiento de un Robot Móvil Tipo Diferencial Robot link.
- [14] Colombia, V., Montoya, J., Rios, A., MODELO CINEMÁTICO DE UN ROBOT MÓVIL TIPO DIFERENCIAL Y NAVEGACIÓN A PARTIR DE LA ESTIMACIÓN ODOMÉTRICA (2009). (pp. 191–196). link.
- [15] Seguimiento de Trayectoria Mediante la Solución del Error Lateral en un Robot Tipo Diferencial. (s.f.). link.
- [16] Mariana. (2024, Enero 4). Control robusto. Fisicotrónica. link.
- [17] Castellanos Pérez, M., Medina, A., Álvaro Hernández, S., Lester, S., Maza, A., León, I., y Revisores, O. (s.f.). Instituto Tecnológico de Tuxtla Gutiérrez Reporte Final Residencia Profesional: Desarrollo y Aplicación de Algoritmos de Cooperación en Robots Móviles. link.
- [18] Cortés, U., Castañeda, A., Benítez, A., y Díaz, A. (2015). Control de Movimiento de un Robot Móvil Tipo Diferencial Robot link.
- [19] Seguimiento de Trayectoria Mediante la Solución del Error Lateral en un Robot Tipo Diferencial. (s.f.). link.
- [20] Mariana. (2024, Enero 4). Control robusto. Fisicotrónica. link.
- [21] NVIDIA Jetson Nano. (s.f.). NVIDIA. link.
- [22] OpenCV. (2024, Mayo 1). OpenCV - Open Computer Vision Library. link.
- [23] NVIDIA CUDA toolkit - free tools and training. (s.f.). NVIDIA Developer. link.
- [24] Instala OpenCV 4.5 en NVIDIA Jetson Nano — Configura una cámara para Jetson Nano. (2024, Marzo 4). link.
- [25] Greyrat, R. (2022, July 5). ¿Cómo detectar formas en imágenes en Python usando OpenCV? – Barcelona Geeks. link.
- [26] Administrador. (2020, Febrero 11). Detección de colores en OpenCV – Python (En 4 pasos). OMES. link.
- [27] Procesamiento Digital de Imágenes. (Agosto 2006). link.
- [28] ¿Qué es un robot seguidor de línea? Spiegato. (2021, Julio 12). link.
- [29] A. Pinargote, B., Jean, D., y García, M. (s.f.). UNIVERSIDAD POLITÉCNICA SALESIANA SEDE GUAYAQUIL. Diseño e Implementación de un Robot Seguidor de Línea mediante Visión Artificial. link.
- [30] Santos, D., Dallos, L., Gaona García, P. A. (2020). Algoritmos de rastreo de movimiento utilizando técnicas de inteligencia artificial y machine learning. Informacion Tecnologica. link.
- [31] ManchesterRoboticsLtd. (s. f.). GitHub - ManchesterRoboticsLtd/TE3002B-2024: Intelligent Robotics Implementation. GitHub. link.
- [32] Tutorial de red neuronal convolucional (CNN) en Python con TensorFlow. (2020, Diciembre 13). ICHI.PRO. link.
- [33] Ultralytics YOLOv8 (s. f.). IA de Visión Avanzada. link.
- [34] RoboFlow. (s. f.). Roboflow: Computer vision tools for developers and enterprises. link.

APPENDIX A

REPOSITORIO DEL PROYECTO

Link de GitHub del proyecto: [link](#).

APPENDIX B

PRESENTACIÓN MANCHESTERROBOTICS

Link de la presentación realizada para Manchester Robotics [link](#).

APPENDIX C

VIDEO DEMOSTRATIVO

Link al video demostrativo realizado para Manchester Robotics [link](#).