

Implementación de PuzzleBot Autónomo con Visión Computacional y Redes Neuronales

Esteban Padilla Cerdio¹, Karen Cebreros López¹, Andrea González Arredondo¹, Naomi Estefanía Nieto Vega¹, Aranza Leal Aguirre¹

¹Escuela de Ingeniería y Ciencias del Tecnológico de Monterrey Campus Querétaro, México.

Este trabajo fue apoyado en parte por Manchester Robotics en colaboración con el Tecnológico de Monterrey Campus Querétaro.

RESUMEN Este proyecto consiste en el desarrollo de un robot móvil autónomo equipado con un sistema de visión computacional y control inteligente, en el cual, nuestro objetivo principal es diseñar, armar, simular e implementar en físico un prototipo de robot llamado PuzzleBot, proporcionado por Manchester Robotics, que es capaz de navegar de manera autónoma en un entorno a escala, en el que se puede encontrar con diferentes obstáculos curvas, cruces, señales de tránsito y semáforos. Para el desarrollo de este proyecto se utilizó ROS (Robotics Operating System) y un kit de desarrollo, en particular una Jetson Nano de NVIDIA con capacidades para realizar operaciones de inteligencia artificial con redes neuronales para el reconocimiento de señales y detección del entorno. Finalmente, gracias a este proyecto se logró la implementación de un prototipo funcional que demuestra el potencial de las aplicaciones de la robótica y la inteligencia artificial en la solución de problemas del mundo actual y desafíos tecnológicos.

PALABRAS CLAVE control inteligente, inteligencia artificial, Jetson Nano, Manchester Robotics, navegación autónoma, NVIDIA, redes neuronales, ROS, visión computacional.

I. INTRODUCCIÓN

El presente proyecto se enfoca principalmente en el área de la robótica móvil autónoma y la inteligencia artificial en conjunto con otras herramientas útiles. Nuestro objetivo principal consiste en armar el robot PuzzleBot de Manchester Robotics e implementar la programación para que sea capaz de llevar a cabo tareas complejas, de forma autónoma con ayuda de visión computacional y control inteligente.

Para lograr el proyecto empleamos una combinación de algoritmos y tecnologías diversas, que en conjunto nos sirvieron para lograr los requerimientos del proyecto. Dicho proyecto se dividió en secciones de desarrollo, las cuales son:

a) Ensamblaje del PuzzleBot móvil autónomo

En esta primera etapa se llevó a cabo la construcción y ensamblaje de las piezas del robot utilizando el kit de desarrollo y componentes de hardware necesarios para que nuestro robot funcione correctamente, se realizó el cableado y la interconexión de los dispositivos electrónicos necesarios.

b) Implementación de la visión computacional y control inteligente

En esta segunda etapa realizamos la implementación de los algoritmos de visión computacional, como el filtro gaussiano, el threshold binario y la obtención de contornos. Asimismo, se implementó la utilización de redes neuronales y algoritmos de control PID para lograr un control inteligente en nuestro robot.

c) Simulación y validación del robot

En esta tercera etapa, se llevó a cabo la simulación de nuestro PuzzleBot para comprobar el funcionamiento de los algoritmos implementados en la etapa anterior y su funcionalidad en un entorno simulado. Además, se detallarán los resultados obtenidos para compararlos con los objetivos planteados.

d) Implementación y pruebas física del robot

Por último, en esta etapa se verificará el funcionamiento de los algoritmos implementados, como el seguidor de línea, el ubicador de cruces y el detector de señales y tráfico. Asimismo, se analizará la precisión y eficacia de cada uno de estos en relación con los objetivos del proyecto.

II. OBJETIVOS

1. Construir un robot móvil autónomo que pueda navegar de forma autónoma en un entorno real, utilizando técnicas de visión, inteligencia artificial y control inteligente.
2. Implementar los algoritmos de visión computacional para el procesamiento de las señales y detección de elementos, como líneas de seguimiento, cruces y señales de tráfico.
3. Entrenar y utilizar redes neuronales, en particular el modelo YOLO v5, para la detección y reconocimiento de objetos para permitir que el

- robot tome decisiones y realice movimientos precisos en función de los datos que va procesando.
4. Desarrollar los algoritmos de control inteligente para permitir que el robot tome decisiones y realice los movimientos de forma precisa y fluida.
5. Implementar una máquina de estados que integre los diferentes módulos y algoritmos para permitir la coordinación y el cambio del comportamiento dependiendo de la situación.
6. Realizar pruebas físicas en un entorno físico y simulado para evaluar la capacidad del robot en la navegación autónoma.

III. DESARROLLO

En el campo de la robótica móvil, existen diversas soluciones a distintas problemáticas que han demostrado ser efectivas, por lo que, en esta sección expondremos un análisis del procedimiento para dichos retos, comenzando por el estado del arte, para el cual presentaremos proyectos recientes similares a la implementación de nuestro reto.

A. ESTADO DEL ARTE

La robótica móvil ha experimentado avances muy significativos en las últimas décadas, impulsados principalmente por soluciones comunes utilizadas en el campo. En esta sección, exploraremos algunas de las soluciones más comunes en la robótica móvil, analizaremos sus ventajas y desventajas, y compararemos cómo nuestro propio proyecto se asemeja a estas soluciones.

Entre las soluciones a las problemáticas mencionadas anteriormente, contamos con el primer ejemplo, el Tesla Autopilot.

01. PROYECTO 1: TESLA AUTOPILOT

Esta implementación básicamente es un sistema de asistencia a la conducción desarrollado por Tesla, Inc. para sus vehículos eléctricos que ofrece reconocimiento de señales y objetos en un entorno urbano para los pasajeros, como se muestra en la figura 1. Este sistema utiliza una combinación de cámaras, sensores ultrasónicos y radares para detectar y responder a los objetos y las condiciones de la carretera. [3]

Es importante recalcar que el Autopilot es únicamente un sistema de asistencia al conductor, y no necesariamente un sistema de conducción autónoma total. Sin embargo, ofrece varias funciones útiles para los conductores, como el control de crucero adaptativo para ajustar la velocidad del vehículo y mantener una distancia segura respecto a los demás. También incluye funciones de cambio de carril y estacionamiento automático, y la capacidad de responder a las señales de tráfico y condiciones del entorno.

Las principales ventajas del Tesla Autopilot principalmente son:

Comodidad

Gracias a la navegación autónoma del Tesla Autopilot, se puede lograr que la conducción sea más relajada y cómoda, especialmente en viajes largos o en tráfico congestionado. El control de crucero y la dirección asistida pueden ayudar a reducir la fatiga del conductor al manejar los pedales en determinadas situaciones.

Mejora de la seguridad

En el ámbito de seguridad, el Tesla Autopilot puede ayudar a evitar accidentes al mantener una distancia segura con respecto a otros vehículos, ya que mantiene el automóvil centrado en su carril y responde rápidamente a las condiciones de la carretera.

Por otro lado, las desventajas de este sistema son:

Limitaciones y dependencia del conductor

Debido a que no es un sistema de conducción autónoma total, el conductor debe permanecer atento y preparado para tomar el control en cualquier momento. Es por ello que, una de las desventajas sería crear una dependencia excesiva del sistema y la falta de atención. Esto puede ser peligroso y llevar a situaciones de riesgo.

Condiciones limitadas

El Tesla Autopilot puede tener ciertas dificultades en situaciones y condiciones de la carretera, como por ejemplo, en obras viales, condiciones climatológicas adversas, carreteras mal señaladas o situaciones de tráfico complejas. En estos casos el sistema puede requerir la intervención del conductor o ser desactivado temporalmente.

Este sistema difiere de nuestro proyecto de manera que, el PuzzleBot es una implementación, hasta cierto punto, primitiva, de la navegación autónoma, ya que tenemos una vialidad definida y no se pone en riesgo la vida de ninguna persona. Nuestro proyecto es una experiencia de acercamiento a cómo se implementa un proyecto de conducción autónoma en la vida real, sin embargo, la mecánica y el hardware implementado son ligeramente diferentes a los del Tesla Autopilot, en el cual, su objetivo principal es mejorar la seguridad y experiencia de conducción en vehículos de carretera, como se muestra en la Figura 1.



FIGURE 1. Demostración de la funcionalidad del Tesla Autopilot en un entorno urbano. Obtenido de: https://www.tesla.com/es_MX/autopilot

02. PROYECTO 2: ROBORACE: COMPETICIÓN DE VEHÍCULOS AUTÓNOMOS

Roborace es una competición con coches autónomos y eléctricos. Creado por el empresario en serie Denis Sverdlov, Roborace se anunció oficialmente en 2015, en asociación con la Fórmula E. Sverdlov cree que el futuro de los automóviles es eléctrico y autónomo. Al impulsar la tecnología en las carreras, se volverá más común, más eficiente y más probable que obtenga una adopción generalizada. Sverdlov planea tener diez autos en funcionamiento que representan a diferentes equipos, probablemente marcas automotrices de alta potencia como en la Fórmula 1. [4]

Para que todo esto fuese posible se instaló una poderosa IA en los autos, que están equipados con diferentes sensores para darle un tipo de conciencia en tiempo real mientras se mueve alrededor de la pista. Hay dos radares, que se usan para detectar objetos en la distancia y LiDAR, que dispara haces de luz en múltiples direcciones antes de usar los reflejos para determinar dónde están los objetos circundantes. Esto se complementa con 18 sensores ultrasónicos que se utilizan para la detección a corta distancia, dos sensores de velocidad y seis cámaras que utilizan la visión artificial para ofrecer una vista de 360° alrededor del automóvil. También hay un Sistema satelital de navegación (GNSS), una forma de tecnología GPS.

Con Roborace, la sofisticación del software marcará la diferencia entre quién gana y quién pierde. En teoría debería significar que la IA en los autos se vuelve más inteligente y flexible a medida que el gran premio transurre de carrera en carrera. En la figura 2 podemos observar un ejemplo de los coches que participan en esta carrera. Por otro lado, las principales ventajas de Roborace son:

Promoción de la seguridad vial

Roborace busca impulsar avances en la seguridad vial. La competencia permite probar y perfeccionar sistemas de detección de obstáculos, respuesta rápida a situaciones imprevistas y toma de decisiones autónomas, lo que puede eventualmente traducirse en mejoras en la seguridad de los vehículos de conducción autónoma en las carreteras convencionales. [7]

Impulso a la innovación tecnológica

Roborace fomenta el desarrollo y la mejora de la tecnología de conducción autónoma. Los equipos participantes tienen la oportunidad de aplicar soluciones innovadoras y avanzadas en la programación y control de vehículos autónomos, lo que impulsa la investigación y el avance de la tecnología en este campo. [7]

Y las desventajas serían:

Dependencia de la tecnología

Roborace se basa en la tecnología de conducción autónoma, lo que implica una dependencia significativa de sistemas y algoritmos complejos. Si ocurren fallas o errores en la tecnología durante una carrera, esto puede afectar negativamente la experiencia de los espectadores y la credibilidad de la competencia.[8]

Costos elevados

Participar en Roborace y desarrollar vehículos autónomos de alto rendimiento puede implicar costos significativos para los equipos y las organizaciones involucradas. Esto puede limitar la accesibilidad y la participación de ciertos equipos y limitar la diversidad y la competencia en la competencia. [8]

Esta solución difiere de nuestra implementación, principalmente porque la implementación de nuestro PuzzleBot es ligeramente más a escala y los costos de manufactura son muy distintos, ya que este consiste de un kit desarrollado para que podamos empezar a implementar las soluciones de software sin realizar un diseño de hardware que vaya a ser compatible o que pueda funcionar con el software que desarrollamos.



FIGURE 2. Imagen ilustrativa de uno de los coches que participan en la competencia Roborace. Obtenido de: <https://www.revistacar.es/magazine-noticia/roborate-onboard-coche-a-utonomo>

B. IMPLEMENTACIÓN, MATERIALES Y MÉTODOS

Para este reto, inicialmente nos proporcionaron un PuzzleBot desarmado con todos los componentes electrónicos y de hardware necesarios para tener el robot en el que instalaremos todo el software desarrollado, que es el principal reto de este proyecto. Como podemos observar en la figura 3.

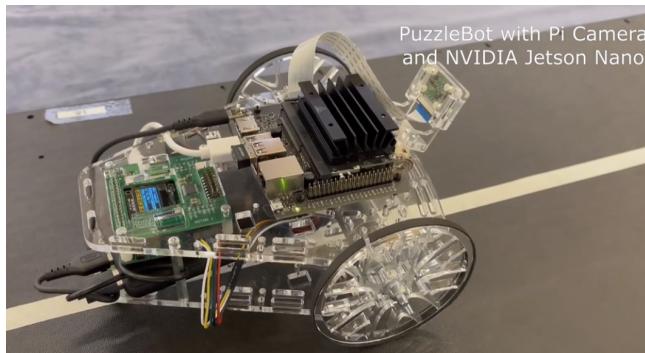


FIGURE 3. En esta figura podemos observar el PuzzleBot con todos sus componentes armados. Imagen obtenida de: <https://manchester-robotics.com/PuzzleBot/>

Para la implementación del software de nuestro PuzzleBot básicamente creamos varios nodos, a continuación explicaremos la funcionalidad que cada uno tiene para realizar el control de nuestro robot:

Line follower

El nodo "Line Follower" es un algoritmo diseñado para guiar a un robot en función de una línea en una pista. El proceso comienza capturando una imagen en tiempo real mediante una cámara. Luego, se redimensiona la imagen a 200x200 píxeles y se toma la parte inferior de la imagen, ya que solo nos interesa la pista que se encuentra frente al robot, descartando el resto de la imagen.

A continuación, se convierte la imagen a blanco y negro y se aplica un filtro gaussiano con un kernel de 5x5 para suavizar la imagen. Luego, se emplea una combinación de erosión y dilatación utilizando un kernel de 5x5 para eliminar los detalles pequeños y obtener una representación más clara de la línea.

Después, se aplica un filtro binario invertido, que convierte en blanco todo lo que era negro y viceversa. Esto resalta la línea como una mancha blanca en la imagen, siendo el único elemento destacado. Seguidamente, se utiliza la función de detección de contornos proporcionada por CV2 (OpenCV) y se eliminan los contornos que son demasiado pequeños.

A continuación, se obtienen los centroides de los contornos más grandes. Al inicio, se busca el centroide más cercano al centro de la imagen y, a partir de ahí, se utiliza un algoritmo de seguimiento (tracking) para rastrear ese centroide. Esto permite seguir el mismo centroide incluso si se mueve o aparecen más contornos en la imagen.

La posición del centroide con respecto al centro de la imagen se publica y se utiliza como entrada para un controlador PID. El controlador PID determina una velocidad angular que se envía al robot. El robot utiliza esta velocidad angular, junto con una velocidad lineal constante, para seguir la línea de manera precisa y continua.

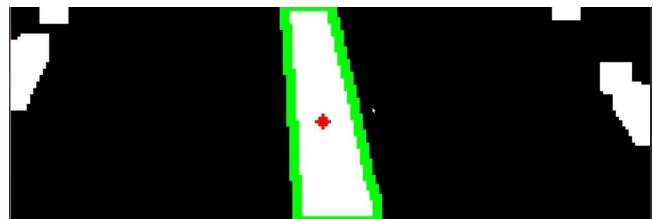


FIGURE 4. En esta imagen podemos observar lo que nuestro nodo del seguidor de línea observa en la pista, identifica la línea y su centro para proceder a seguir la trayectoria marcada.

Crossroad detector

Ahora bien, el nodo "Crossroad Detector" es un algoritmo creado para detectar y tomar decisiones en un cruce de caminos basado en una imagen capturada por una cámara. El proceso comienza recibiendo la imagen directamente de la cámara y redimensionándola a 500x500 píxeles.

A continuación, se toma la mitad inferior de la imagen, ya que no nos interesa la parte superior para esta aplicación. La imagen se convierte a blanco y negro y se aplica un filtro gaussiano con un kernel de 5x5 para suavizarla.

Luego, se utiliza una combinación de erosión y dilatación con un kernel de 5x5 para eliminar los detalles pequeños y obtener una representación más clara. Se aplica un filtro

binario invertido, que convierte en blanco todo lo que era negro y viceversa.

Este proceso resalta los cuadritos del cruce como manchas blancas en la imagen. A continuación, se obtienen los contornos y se eliminan aquellos que son demasiado pequeños, demasiado altos, demasiado anchos o que no tienen forma de rectángulo.

Posteriormente, se obtienen los centroides de todos los cuadritos. Se utiliza un algoritmo de regresión lineal para encontrar todas las posibles líneas que conecten 4 o más puntos. De todas estas líneas, se selecciona aquella que esté más alineada con el eje horizontal.

La posición de esta línea se envía al robot. Si el robot detecta que la línea se acerca, cambia del estado de "Follow Line" al estado de "Crossing Road". Dependiendo de la señal de tráfico, el robot tomará la decisión de dar la vuelta o continuar recto en el cruce.

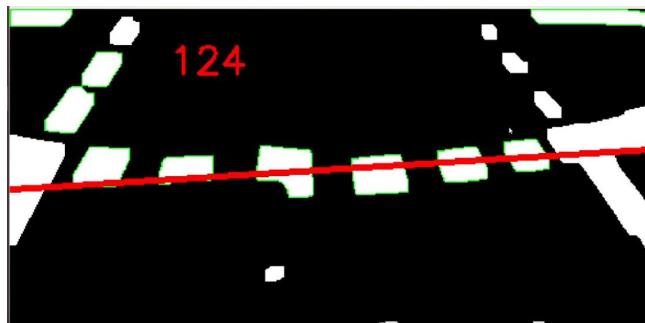


FIGURE 5. En esta imagen podemos observar lo que el nodo de crossroad detector observa en la pista, de tal manera que comprobamos que realiza el funcionamiento correcto, en este caso, identificar los cuadritos del cruce. El numerito que observamos en rojo es la altura de la línea, si pasa de 80 pasamos al estado de Crossing Line.

Sign detector

El sistema recibe la matriz de resultados generada por YOLO, que contiene información sobre los objetos detectados en una imagen. A partir de esta matriz, se realiza un proceso de filtrado para identificar y seleccionar únicamente las señales de tráfico, descartando las luces.

Una vez identificadas las señales, se obtiene el alto y el ancho de cada una, lo que permite calcular el área de cada señal. Se procede a eliminar aquellas señales que tienen un área demasiado pequeña, ya que esto indica que se encuentran a una distancia considerable y no requieren atención inmediata.

A continuación, se determina la señal más cercana en función de la distancia. Esta señal se publica como un mensaje personalizado denominado "Detected Object" (Objeto Detectado), el cual contiene un campo de texto

"name" (nombre) que indica el tipo de señal detectada, y un valor numérico "área" que representa el área de la señal detectada.

De esta manera, el sistema realiza un procesamiento de la matriz de resultados de YOLO para identificar y reportar la señal de tráfico más cercana, proporcionando información útil sobre el tipo de señal y su área de ocupación.

Traffic light detector

El sistema recibe la matriz de resultados generada por YOLO, la cual contiene información sobre los objetos detectados en una imagen. En este caso, se realiza un proceso de filtrado para identificar y seleccionar exclusivamente las luces de tráfico, descartando las señales.

Una vez identificadas las luces, se obtiene el alto y el ancho de cada una, lo que permite calcular el área correspondiente. Se procede a eliminar aquellas luces que tengan un área demasiado pequeña, ya que esto indica que se encuentran a una distancia considerable y no requieren atención inmediata.

Adicionalmente, se aplican criterios específicos de confianza para las luces de diferentes colores. Para las luces rojas y amarillas, se eliminan aquellas que tengan una confianza menor al 70%. En el caso de las luces verdes, donde el modelo de detección puede tener menor precisión, se establece una confianza mínima del 40% para considerarlas válidas.

Luego de aplicar estos filtros, se obtiene la luz de mayor confianza entre las que quedan. Esta luz se publica como un objeto detectado en el mensaje llamado "Detected Object", proporcionando información relevante sobre el tipo de luz de tráfico y su nivel de confianza.

De esta manera, el sistema procesa la matriz de resultados de YOLO, identificando y reportando la luz de tráfico más relevante y confiable, considerando tanto su área como la confianza asociada, lo cual resulta útil para la detección precisa de las luces de tráfico en una imagen.

YOLO (Nodo y entrenamiento)

Para el entrenamiento de la red utilizamos YOLO, en este caso, se utilizó Transfer Learning como enfoque, basado en el modelo YOLO v5 Nano. Se trabajó con un conjunto de datos que constaba de 4,350 imágenes de tamaño 224x224, previamente categorizadas mediante bounding boxes.

El modelo implementado contaba con 214 capas y un total de 1,776,094 parámetros con gradiente, lo que permitió un aprendizaje profundo y detallado. Durante el proceso de

entrenamiento, se realizaron 500 épocas, utilizando un batch size de 64 imágenes.

El entrenamiento se llevó a cabo utilizando una tarjeta gráfica Nvidia RTX 3070, y el tiempo total requerido fue de aproximadamente una hora. Al finalizar el proceso, se logró obtener una precisión del **95% en la detección de objetos**.

Una modificación importante realizada en los hiperparámetros fue la configuración de "Left Right Mirror" (Espejo Izquierda-Derecha), con el propósito de evitar que el modelo categorizará imágenes de izquierda y derecha como si fueran idénticas. Esta modificación permitió una mayor precisión y confiabilidad en la clasificación de objetos.

```
# frame 1
  1769 models.common.Conv [1, 16, 9, 2, 2]
  4886 models.common.Conv [1, 16, 9, 2, 2]
  18089 models.common.Conv [1, 16, 9, 2, 2]
  18189 models.common.Conv [1, 16, 9, 2, 2]
  73984 models.common.Conv [1, 16, 9, 2, 2]
  295424 models.common.Conv [1, 16, 9, 2, 2]
  164886 models.common.SPP [1, 16, 9, 2, 2]
  38864 models.common.Conv [1, 16, 9, 2, 2]
  1482 torch.nn.modules.upsampling.Upsample [none, 2, 'nearest']

[-1, 0] 98000 models.common.Conv [1, 16, 9, 2, 2]
  4329 models.common.Conv [1, 16, 9, 2, 2]
  8 torch.nn.modules.upsampling.Upsample [none, 2, 'nearest']
  22932 models.common.C3 [1, 16, 9, 2, 2]
  22932 models.common.C3 [1, 16, 9, 2, 2]
  22932 models.common.Concat [1, 16, 9, 2, 2]
  74496 models.common.Conv [1, 16, 9, 2, 2]
  147732 models.common.Conv [1, 16, 9, 2, 2]
  294480 models.common.C3 [1, 16, 9, 2, 2]
  17, 20, 23 1 [1, 16, 9, 2, 2]
  2378894 parameters, 3778894 gradients, 4,3 GByte

Model summary: 214 layers, 3778894 parameters, 3778894 gradients, 4,3 GByte
```

FIGURE 6. En esta imagen podemos observar la estructura de YOLO, que básicamente consiste en varias capas convolucionales y una capa final lineal para detectar la clase.

Por otro lado, para el nodo de YOLO se utiliza el framework YOLO v5 de Ultralytics, implementado en Python 3.8. El modelo entrenado se carga en formato ONNX, el cual ha sido optimizado para ejecutarse en la unidad central de procesamiento (CPU). Además, se utilizan funciones de codificación y decodificación de imágenes de ROS (Robot Operating System) debido a que CVBridge no es compatible con Python 3.

El proceso comienza al recibir la imagen de la cámara, la cual se transforma en un arreglo NumPy para su posterior procesamiento. A continuación, se realiza la inferencia utilizando el modelo cargado. En condiciones normales, la latencia de la inferencia es de 0.1 segundos cuando el nodo se ejecuta de forma aislada, y de 0.3 segundos cuando se combina con otros nodos.

Se obtienen todos los objetos presentes en la imagen junto con sus bounding boxes. Posteriormente, se realiza un filtrado aplicando un umbral de confianza para eliminar aquellos objetos que no cumplan con este criterio.

Con el objetivo de realizar un seguimiento preciso de los objetos y evitar la presencia de objetos "fantasma" que aparezcan o desaparezcan en los frames, se utiliza el centro de cada objeto para realizar un tracking. Si un objeto se encuentra presente en tres frames consecutivos, se considera como un objeto real, de lo contrario, se clasifica como un objeto fantasma.

Finalmente, se dibujan todos los objetos detectados en la imagen y se envía la matriz resultante como un tópico para ser recibida por los detectores de señales y luces. Este proceso permite compartir la información de detección con otros módulos o sistemas que requieran identificar y trabajar con estos objetos específicos en tiempo real.

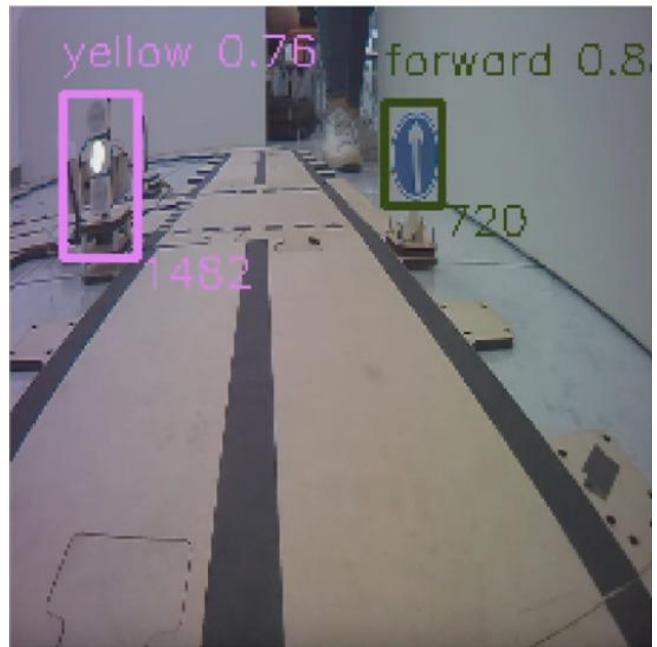


FIGURE 7. En esta imagen podemos observar lo que nuestro nodo y entrenamiento de YOLO identifican a través de la cámara del PuzzleBot.

Máquina de estados

La implementación de la máquina de estados es fundamental en este proyecto. La máquina de estados se encarga de recibir la información proveniente del detector de señales y el detector de luces, y toma decisiones en base a los datos recibidos. Se establece un threshold de tamaño para determinar cuándo se debe prestar atención a las señales, evitando reaccionar a objetos demasiado pequeños que puedan encontrarse a una distancia considerable.

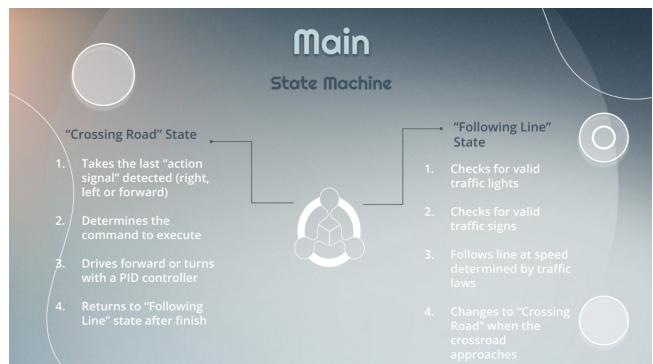


FIGURE 8. En esta figura podemos observar una representación gráfica de la máquina de estados que sigue nuestro robot.

El estado inicial de la máquina de estados es el "Line Follower" (seguidor de línea). En este estado, si no se detecta ninguna señal ni luz, el robot continúa avanzando de manera normal. Si se detecta una luz verde, también continúa avanzando sin cambios. Sin embargo, si se detecta una luz roja, el robot se detiene hasta que dicha luz deje de estar presente en la imagen. En caso de detectar una luz amarilla, el robot disminuye su velocidad a la mitad hasta que la luz amarilla desaparezca. Si se recibe la señal "Give Way" (ceda el paso), el robot disminuye su velocidad a la mitad durante 5 segundos. En caso de detectar la señal de "road work" (obras en la vía), el robot disminuye su velocidad a la mitad durante 2 segundos. Si se recibe la señal "stop" (alto), el robot se detiene durante 10 segundos. Por otro lado, si se detecta una señal de giro o de seguir adelante, el robot no cambia su velocidad actual, pero guarda esta información para futuras decisiones.

Cuando se acerca a una intersección, el robot pasa al estado "Crossing Road" (cruzando la vía) y deja de prestar atención al estado "Line Follower". En este nuevo estado, si la última señal detectada fue una señal de "seguir adelante", el robot avanza una distancia fija calculada utilizando su velocidad lineal y un intervalo de tiempo (DT). Si la última señal detectada fue una señal de "giro", el robot avanza una distancia corta, gira un ángulo utilizando un controlador PID y luego avanza nuevamente una distancia corta. Una vez que se completa el comando de cruce de la intersección, el robot regresa nuevamente al estado "Line Follower" para continuar siguiendo la línea y respondiendo a las señales y luces detectadas.

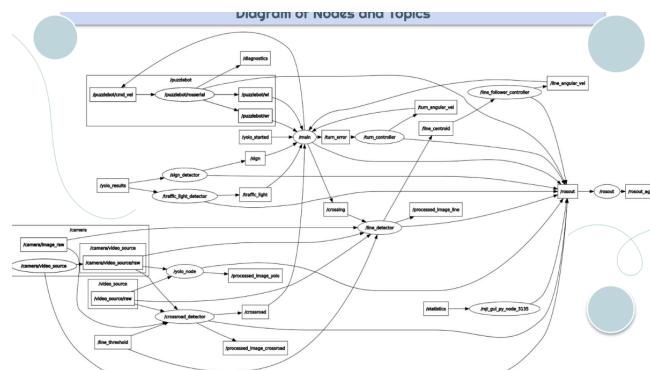


FIGURE 9. En esta figura podemos observar todos los nodos y tópicos que conforman nuestro proyecto en ROS.

Para la parte de materiales, básicamente consisten en una Jetson Nano de NVIDIA para la implementación de los algoritmos de visión e inteligencia artificial en ella. También contamos con una Hacker Board para la interconexión de la Jetson con los motores DC y la demás circuitería, con esto implementamos los algoritmos de control. Además, utilizamos una Cámara de Raspberry Pi para poder observar el entorno con el PuzzleBot, como se muestra en la figura 10.

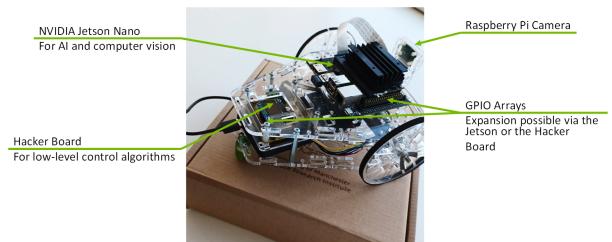


FIGURE 10. En esta figura podemos observar a grandes rasgos los componentes de nuestro robot PuzzleBot.

Ahora bien, para profundizar más en los materiales observemos la figura 11.

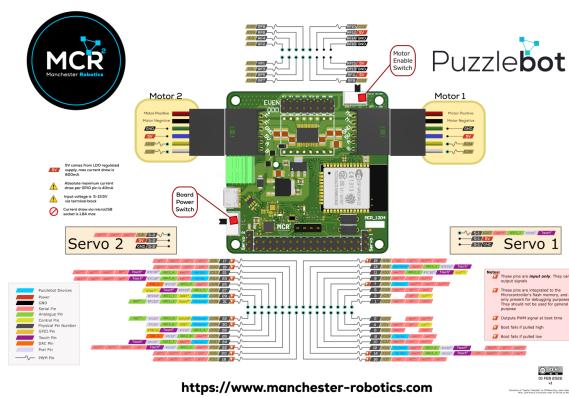


FIGURE 11. En esta figura podemos observar los componentes más a detalle de la HackerBoard que compone al PuzzleBot.

Los materiales que constituyen al PuzzleBot son: NVIDIA Jetson Nano, Raspberry Pi Camera, Hacker Board

La NVIDIA Jetson Nano cuenta con amplias entradas y salidas, desde GPIO hasta CSI, esto nos facilita la conexión de diferentes sensores. Utiliza el sistema operativo de Linux, por lo que puede correr ROS, lo que necesitamos para visión computacional, procesamiento de imágenes, incluso puede utilizar una imagen de una tarjeta SD, lo que la hace fácil de flashear.

IV. SUSTENTABILIDAD DE LOS VEHÍCULOS AUTÓNOMOS

El principal problema que tienen los automóviles de combustión interna, son sus altos niveles de emisión de dióxido de carbono (CO₂). Evidentemente los automóviles autónomos son eléctricos y no producen emisión de gases. Sin embargo, no todos son 100% sostenibles, pues depende del método de recarga de la batería, si es que esta proviene de fuentes limpias. Pero aún así tienen el potencial y la posibilidad de contribuir a la sostenibilidad de varias maneras, aunque también existen posibles desafíos y consideraciones como las siguientes:

Beneficios ambientales

Los vehículos autónomos tienen el potencial de reducir las emisiones de gases de efecto invernadero (GEI) y mejorar la calidad del aire.

Eficiencia energética

La tecnología de conducción autónoma tiene el potencial de optimizar el uso de energía al mejorar el flujo de tráfico y reducir la necesidad de aceleración y frenados excesivos.

Movilidad compartida

Los vehículos autónomos se pueden integrar en los servicios de movilidad compartida. Estos tienen el potencial de reducir la cantidad total de vehículos en la carretera al brindar opciones de transporte eficientes y reducir la necesidad de poseer un automóvil privado. Por lo que esto puede decrementar la congestión, la demanda de estacionamiento y el consumo general de recursos.

Mejoras en la seguridad

La tecnología de conducción autónoma tiene el potencial de reducir los errores humanos, que son una de las principales causas de accidentes.

Desafíos y consideraciones

De igual manera como existen beneficios, también podemos encontrar los desafíos relacionados con la sostenibilidad de los AV. Estos incluyen el impacto ambiental de la fabricación y el mantenimiento de los vehículos autónomos, la fuente de energía utilizada para impulsar vehículos autónomos, la gestión del desperdicio de baterías para vehículos autónomos eléctricos. Y cabe recalcar que el éxito de los AV no depende solo de tecnología, sino también de la planificación de la infraestructura y el comportamiento de los consumidores. Además de que se tiene que considerar si realmente estamos preparados éticamente como sociedad para emigrar hacia esta innovación.

V. RESULTADOS

A través de diferentes pruebas realizadas, se desarrolló un robot móvil capaz de navegar de forma autónoma en un entorno a escala, donde pudo enfrentarse a diferentes situaciones en las cuales tenía que tomar ciertas decisiones.

Los siguientes resultados se lograron gracias a diversos experimentos y pruebas para tener la cantidad suficiente de datos. Al principio, se entrenó la red neuronal con aproximadamente 2500 imágenes, 1,700,000 parámetros y 20 épocas, lo cual no fue suficiente para que el modelo funcionara correctamente, esto generó entre 40% y 60% de

validación para cada una de las categorías. Después, estos datos de entrenamiento provocaban que el robot no pudiera identificar bien las imágenes y confundía las clases. Así que se decidió aumentar el número de datos, por último se tuvieron 4,500 imágenes, 500 épocas y 1,776,094 parámetros, lo cual nos favoreció para obtener una validación del 92% para cada una de las señales.

Para determinar cuál era la máxima velocidad en la que el robot funcionara correctamente, se siguió un procedimiento de prueba y error, la primera velocidad lineal que le dimos fue de 1 m/s, en la que pudo seguir la pista con sus diferentes obstáculos muy bien, por lo que se decidió aumentar la velocidad 0.5 m/s, hasta llegar a 2 m/s que fue la velocidad óptima en la cual se tenía un buen control del robot. Es decir, gracias al control proporcional que se utilizó, podía controlar su trayecto en caso de que existiera un error. Finalmente para las señales, esta velocidad puede ser modificada dependiendo de la señal detectada.

Para el semáforo, cuando la luz roja está encendida, tiene un alto por completo, hasta que detecte el color verde y pueda avanzar, mientras que con el amarillo disminuye su velocidad a 1 m/s hasta que vea el rojo o pase el semáforo.

En cambio, la señal de road work ahead disminuye la velocidad a la mitad por 5 segundos y en la de give way se comporta de la misma forma pero por 2 segundos.

En la pista se puede encontrar cierto cruce, esto significa que tiene que tomar una decisión, en la que puede o no involucrar señales, y en caso de que exista una, se puede encontrar con la señal de vuelta a la izquierda o vuelta a la derecha, en la que al identificarla se le envía una señal de velocidad lineal constante y se aumenta la velocidad angular, para que pueda girar ya sea $\pi/2$ o $-\pi/2$, dependiendo de la orientación de la flecha.

El PuzzleBot es capaz de identificar ocho tipos de señal, y efectuar la condición que está asignada para cada una de ellas, además de que es un seguidor de línea y puede corregir sus errores gracias al control implementado, todo esto lo hace de manera autónoma.

VI. DISCUSIÓN DE RESULTADOS

A continuación se presenta la discusión de resultados:

- Naomi Estefanía Nieto Vega

A continuación, analizaré los resultados y puntos clave relacionados con la implementación de este proyecto. En primer lugar, considero que los algoritmos que implementamos en la sección de materiales y métodos permiten que nuestro robot perciba el entorno a través de diferentes sensores. Principalmente, para la percepción visual, se implementaron algoritmos de detección de señales y luces de semáforo utilizando el modelo YOLO

v5, que fue entrenado previamente con una gran cantidad de imágenes etiquetadas para reconocer las señales en la pista. Con este algoritmo nuestro robot es capaz de identificar señales y su ubicación en la imagen recibida de la cámara. También utilizamos un algoritmo de procesamiento de imágenes para filtrar y “preprocesar” los datos antes de tomar decisiones.

Considero que los principales desafíos en estos aspectos son principalmente la precisión de la detección y calificación de los objetos, así como la velocidad del procesamiento para garantizar una respuesta en tiempo real y que esté a tiempo.

Por otro lado, en cuanto a la navegación se implementó un controlador que utiliza información proporcionada por los sensores y actuadores del robot para lograr el objetivo. El diseño del controlador se basó en la relación entre los sensores, como la cámara y los detectores de señales y luces, y los actuadores, que permiten el movimiento del robot. Se realizaron ajustes en los parámetros y las ganancias del controlador para lograr un rendimiento óptimo y una respuesta adecuada a las observaciones del entorno. La interacción con el entorno se logra a través de las salidas generadas por el controlador, que determinan la velocidad y la dirección del robot en función de la detección de señales y luces, permitiendo reaccionar de manera adecuada ante las situaciones encontradas.

Durante la programación del robot, se enfrentaron varios desafíos significativos. Uno de los principales fue la correcta configuración y calibración de los sensores y actuadores para garantizar su correcto funcionamiento y precisión. Además, la integración de los diferentes módulos y algoritmos en un sistema coherente y sincronizado también presentó dificultades, ya que era necesario establecer una comunicación eficiente entre ellos. Para superar estos desafíos, se realizaron pruebas exhaustivas, ajustes finos y se implementaron mecanismos de sincronización adecuados. Además, se hizo necesario el análisis y depuración detallados de los algoritmos para mejorar su rendimiento y precisión.

• Esteban Padilla Cerdio

La autonomía del robot resultante de este proyecto proviene de los distintos nodos implementados en su programación. El nodo de seguidor de línea, con sus algoritmos de visión computacional, detección de contornos y control PID, le permite seguir el camino que tiene delante. Con algoritmos similares, se logró implementar un detector de cruces que le brindan al robot la oportunidad de tomar acciones de movilidad en el momento adecuado.

Por otro lado, los algoritmos de inteligencia artificial, implementados con el modelo YOLO, y la implementación clásica de una máquina de estados, dotan al robot de la capacidad de tomar decisiones acertadas según lo observado en su entorno. Gracias a estos nodos y

algoritmos, el robot es capaz de percibir las señalizaciones y luces, y reaccionar correctamente acorde a ellas.

Junto con estas soluciones, llegaron los retos correspondientes. Principalmente, las limitaciones de hardware hacen que los detectores de entornos con base en inteligencia artificial tengan una latencia demasiado alta como para su implementación en un entorno real. El tiempo de reacción elevado provoca que sea necesario bajar la velocidad del sistema y tomar decisiones con mayor lentitud. Esto, en un entorno de vialidad a escala natural, no es apropiado ni efectivo. Los parámetros y ganancias de control y los parámetros de visión e IA, así como el modelo de YOLO utilizado (su versión Nano), fueron seleccionados a partir de estas limitaciones.

A pesar de los distintos retos enfrentados durante la solución de este reto, el equipo fue capaz de encontrar soluciones acertadas. Para los retos de visión, como la confusión del sistema ante distintas líneas en el suelo o la correcta identificación de cruces, se implementaron algoritmos como un sistema de tracking de centroides y un sistema de regresión lineal, respectivamente. Por otro lado, los retos del YOLO, como la alta latencia y el ruido ocasionado por objetos “fantasma”, se solucionaron utilizando un modelo optimizado ONNX y un sistema de registro de objetos.

En concreto, los resultados obtenidos por el equipo fueron adecuados y más que suficientes para demostrar la correcta compleción del reto. A pesar de que aún sería necesario realizar modificaciones para implementarlo en un entorno real, los objetivos establecidos fueron correctamente atacados y vencidos.

• Karen Cebreros López

Para la versión final de nuestro proyecto, logramos que el PuzzleBot fuera capaz de detectar 8 tipos de señales y tomar decisiones dependiendo a ellas, para que junto con el algoritmo del seguidor de línea, se lograra el manejo autónomo con una alta efectividad.

Esto se logró a través de la implementación de diferentes algoritmos y el uso de diversas herramientas, que a lo largo de los entregables del reto, nos fueron de gran utilidad para la construcción del reto final.

Los algoritmos que se implementaron, tanto para el seguidor de líneas, como para la detección de cruces, ambos trabajaron con un threshold binario, con la finalidad de trabajar exclusivamente con las líneas negras que ante la cámara del PuzzleBot se veían blancas. Esto junto a la detección de contornos y la obtención de centroides, fue como fuimos capaces de la construcción de estos dos estados, que posteriormente conformaron la máquina de estados dentro de nuestro programa principal, para lo que es el movimiento de nuestro robot.

Luego, a través del entrenamiento de una red neuronal convolucional implementada con YOLO v5 y ejecutada con ONNX; que si bien esto no salió a la primera, poco a poco fuimos mejorando nuestros datos de entrada y parámetros de la red, con los que logramos obtener una efectividad mayor de 90%; lo suficientemente confiable para detectar todas las señales y luces de los semáforos.

Cabe mencionar que utilizamos ONNX con el fin de optimizar la ejecución en la GPU. Pues al final decidimos optar por esta opción, ya que tuvimos muchos inconvenientes al intentar utilizar la GPU con Cuda.

Y para los controladores, optamos por utilizar un controlador PID debido a la experiencia con la que ya contamos en estos y además, porque necesitábamos nuevamente trabajar en minimizar el error a través de un sistema de retroalimentación.

Algo interesante que nos pasó aquí y evidentemente era de esperarse, era que siempre teníamos que volver a tunear los parámetros en el robot aún cuando ya en las simulaciones nos saliera bien. Pues evidentemente ante las condiciones del entorno en donde probamos el robot (por ejemplo el material de la pista o el suelo por la fricción), éste iba a actuar diferente.

Cabe mencionar que esto también nos pasaba para los parámetros del seguidor de línea y del cruce, que tenían que ver con la luz para el threshold binario. Pues dependiendo de la intensidad de la luz, o de si ésta era natural o artificial, se necesitaban ajustar estos parámetros para el correcto funcionamiento de nuestro robot.

Cada etapa de este reto presentó sus dificultades. Sin embargo, considero que gracias a las numerosas pruebas que realizamos, pudimos seleccionar las herramientas que consideramos más óptimas, dependiendo de las características de nuestro PuzzleBot, con la finalidad de no sólo hacerlo eficiente, sino también eficaz.

Así mismo, siento que de los mayores desafíos con los que nos enfrentamos durante el desarrollo del reto, fue juntar todos los algoritmos y herramientas para que trabajaran en conjunto. Pues más de una vez nos encontramos con problemas de compatibilidad de versiones, incoherencias entre tópicos y hasta errores para la funcionalidad de componentes, que en ocasiones nos llevaron a tener que volver a instalar todo de cero. Sin embargo, creo que todos estos errores fueron los que nos permitieron tener un mejor entendimiento tanto del hardware, como del software, que debíamos tener para que estos pudieran trabajar en conjunto y así pudiéramos completar exitosamente el reto.

- **Andrea González Arredondo**

A lo largo de la implementación del PuzzleBot, hubo diferentes complicaciones, principalmente fue que la versión de CUDA no fue compatible, por lo que no se pudo usar el GPU de la Jetson y eso nos causaba una latencia

muy alta, debido a esto, fue que se decidió usar un modelo en formato ONNX, que está optimizado para correr en CPU. De esta manera fue que se logró una latencia de 0.3 segundos ya corriendo todos los nodos. Mientras que para los resultados de la red neuronal tiene una validación alta, gracias a la cantidad de datos que obtuvimos, pero como se mencionó, al principio no fue así, ya que teníamos poca validación, por la falta de imágenes y épocas, para resolverlo se tuvo que volver a entrenar nuestro modelo con más datos, y así obtuvimos una validación arriba de 90%, lo cual fue muy bueno.

Por otro lado, para determinar el valor de la velocidad máxima, se hicieron pruebas de ensayo y error, de esta manera se pudo saber cuánto podía soportar el PuzzleBot para poder recorrer la pista con éxito.

Para los materiales utilizados, fueron un reto, ya que existían diferentes limitaciones, pero a partir de ellas se pudo desarrollar los algoritmos necesarios para sacarle el máximo provecho. Lo que se usó principalmente fue ROS que nos permite conectar nuestros códigos por medio de nodos y tópicos para que funcionen en sincronía, lo que permite una mayor eficiencia. El lenguaje principal que se usó fue Python, que es el más utilizado para redes neuronales, por la gran ventaja que genera las librerías que contiene, esto nos permitió poder limpiar la imagen que estaba recibiendo a partir de diferentes funciones. Todos estos algoritmos y materiales dotaron de autonomía a el PuzzleBot y le dieron capacidades de percepción, navegación e interacción.

El robot logró ser un seguidor de línea, para esto usamos un filtro que convierte el color negro, en blanco y todos los demás colores en negro, para que así identificara principalmente las líneas, después sacamos la línea más cercana al centro de lo que ve la cámara para así poder seguirla. Posteriormente se enfocaba en detectar las señales y las luces del semáforo, para que esto sucediera usamos Yolov5 de Ultralytics y Python3.8, que nos permitió obtener un modelo óptimo para su funcionamiento. Entre los principales retos que esto nos causó fue que tenía la probabilidad de confundir la señal de vuelta a la izquierda con la de vuelta a la derecha, por lo que para resolverlo se modificó el hiperparámetro de “Left Right Mirror” para que no categorizara la izquierda y la derecha en la misma clase.

El PuzzleBot estaba siendo controlado por un PID, esto nos proporciona un mecanismo de retroalimentación para que lo que detecten nuestros sensores sirva para tener una mejor

precisión, eliminando oscilación y aumentando la eficiencia de nuestros actuadores. La selección de nuestros parámetros y ganancias fue a través de ver cómo se comportaba nuestro PuzzleBot en el entorno físico y así, obtener las ganancias aptas para nuestro controlador. Las salidas generadas por el entorno, provocaba que se tuviera que modificar las ganancias muy a menudo, porque no siempre se tenía la misma cantidad de luz, los colores que había eran diferentes, entre otras cosas. Esto también fue un reto para cada uno de nosotros, además de todo lo anteriormente mencionado, esto se resolvió a base de lógica, consistencia, e investigación, para responder de la manera correcta a cada uno de los contratiempos.

- **Aranza Leal Aguirre**

En el análisis de los resultados y puntos clave de este proyecto, es importante destacar cómo los algoritmos implementados en la sección de materiales y métodos han permitido al robot adquirir capacidades de percepción, navegación e interacción con el entorno. En términos de percepción, se han empleado algoritmos de detección de señales y luces de tráfico utilizando el modelo YOLO v5, previamente entrenado con imágenes etiquetadas para reconocer estas señales. Esto ha permitido al robot identificar las señales presentes en la imagen captada por la cámara. Además, se ha aplicado un algoritmo de procesamiento de imágenes para filtrar y preparar los datos obtenidos antes de tomar decisiones.

En cuanto a la navegación, se ha diseñado un controlador que utiliza la información proveniente de los sensores y actuadores del robot para lograr los objetivos planteados. Este controlador se ha desarrollado considerando la relación entre los sensores, como la cámara y los detectores de señales y luces, y los actuadores que permiten el movimiento del robot. Se han ajustado parámetros y ganancias del controlador para obtener un rendimiento óptimo y una respuesta adecuada a las observaciones del entorno. Las salidas generadas por el controlador determinan la velocidad y dirección del robot en función de la detección de señales y luces, lo que le permite interactuar de manera adecuada con el entorno.

Durante la programación del robot, se han enfrentado diversos desafíos significativos. Uno de ellos ha sido la correcta configuración y calibración de los sensores y actuadores, con el fin de garantizar su adecuado funcionamiento y precisión. Además, la integración de los diferentes módulos y algoritmos en un sistema coherente y sincronizado ha presentado dificultades, ya que se ha requerido establecer una comunicación eficiente entre ellos. Para superar estos desafíos, se han llevado a cabo pruebas exhaustivas, ajustes precisos y la implementación de mecanismos de sincronización adecuados. Asimismo, se ha

realizado un análisis detallado y una depuración minuciosa de los algoritmos para mejorar su rendimiento y precisión.

Finalmente, la implementación de los algoritmos descritos en la sección de materiales y métodos ha dotado al robot de habilidades de percepción, navegación e interacción con el entorno. Los algoritmos de detección de señales y luces, así como el controlador desarrollado, han desempeñado un papel fundamental en la toma de decisiones y en la capacidad del robot para interactuar adecuadamente con su entorno. A pesar de los desafíos encontrados durante la programación, se han aplicado estrategias y técnicas para superarlos y mejorar el rendimiento general del robot.

VII. CONCLUSIONES

A continuación se presentan las conclusiones del equipo:

- **Naomi Estefanía Nieto Vega**

Durante el desarrollo de este proyecto, hemos logrado implementar algoritmos que permiten una interacción inteligente entre nuestro robot y el entorno. Los algoritmos de detección de señales y luces de tráfico, basados en el modelo YOLO v5, han demostrado ser efectivos para reconocer y localizar las señales en la imagen captada por la cámara del robot. Además, el controlador diseñado ha permitido que el robot reaccione de manera adecuada ante las observaciones del entorno, ajustando su velocidad y dirección según las señales y luces detectadas. Estos algoritmos han sido fundamentales para dotar al robot de autonomía y capacidades de percepción y navegación.

No obstante, existen áreas de mejora y trabajo a futuro en nuestro proyecto. En primer lugar, se podría buscar mejorar la precisión de los algoritmos de detección, especialmente en situaciones desafiantes como la detección de luces de semáforo en condiciones de baja luminosidad o con ángulos de visión complicados. Esto podría lograrse mediante la recopilación de un conjunto de datos más diverso y exhaustivo para el entrenamiento del modelo, así como la exploración de técnicas de mejora de la detección, como el uso de algoritmos de fusión sensorial.

Además, una función del robot que podría mejorarse es la capacidad de respuesta en tiempo real. Aunque hemos logrado alcanzar una latencia aceptable en la inferencia de los algoritmos, aún existen oportunidades para optimizar el procesamiento y reducir aún más el tiempo de respuesta. Esto podría lograrse mediante la optimización de los algoritmos y el uso de hardware más potente o técnicas de paralelización para acelerar el procesamiento de la información.

- **Esteban Padilla Cerdio**

Los algoritmos y procedimientos utilizados durante el desarrollo de este proyecto, pertenecientes a las áreas de

control clásico, inteligencia artificial y visión computacional, demostraron ser suficientes para implementar un sistema móvil autónomo capaz de desenvolverse en un entorno similar al real. Los métodos de visión fueron útiles a la hora de detectar cruces y líneas, y los modelos de inteligencia artificial fueron herramientas adecuadas para la identificación de señalizaciones de tráfico. Esto, combinado con un control PID sencillo y una máquina de estados, fue suficiente para cumplir con los objetivos establecidos.

No obstante, las limitaciones de Hardware, tales como la resolución y ruido de la cámara, la capacidad limitada de memoria de video de la Jetson NANO y la simplicidad de los actuadores y sensores. demostraron la importancia de tener acceso a los materiales adecuados para implementaciones de este tipo. Estas limitaciones nos obligaron a presentar un prototipo final de baja velocidad de movimiento y alta latencia en los algoritmos de detección. Los parámetros y ganancias de control PID, de detección de objetos y de velocidad de los actuadores fueron establecidas alrededor de estas limitaciones. Por lo tanto, para la emulación de este proyecto en un entorno a escala real, sería necesaria la utilización de herramientas de Hardware con mayor capacidad. Principalmente, con mayor capacidad gráfica, se podría disminuir la latencia del sistema YOLO y mejorar la reacción del robot ante objetos en el entorno.

Fuera de este hecho, los resultados obtenidos son evidencia del éxito de este equipo en el desarrollo del proyecto.

• Karen Cebreros López

Personalmente para concluir, diría que la parte más complicada del proyecto, fue hacer la fusión de todas las partes para lograr el funcionamiento completo del robot, tal y como se nos requería en el reto.

Considero que las herramientas y métodos que decidimos utilizar para la parte de IA y la ejecución del programa (YOLO v5 y ONNX), fueron fundamentales para la última parte del desarrollo. Pues con YOLO v5 pudimos lograr una red con un 92% de precisión y gracias a que se cargó en formato ONNX, pudimos optimizar la ejecución en CPU, obteniendo una latencia muy baja. Cabe mencionar que esto lo hicimos debido a los problemas que tuvimos al intentar utilizar la GPU originalmente para esto.

De igual forma, considero que la funcionalidad del robot se podría mejorar, si se utilizara un mejor hardware en éste, ya que debido a la versión de la Jetson que teníamos, salieron muchos problemas de para la instalación de algunas herramientas (incluyendo algunas versiones de Python, que solucionamos utilizando dos versiones diferentes de éste) y el uso de algunas otras como Cuda para la GPU, como ya lo había mencionado anteriormente.

No obstante, también considero que los resultados que obtuvimos para la versión final de nuestro sistema en el PuzzleBot, fueron bastante buenas y que a pesar de las dificultades encontradas en su desarrollo, logramos encontrar buenas alternativas que no solo nos permitieron cumplir con todos los requisitos del reto, sino que hicieron que éste fuera muy eficiente (comprobado con la latencia tan baja con la que contábamos), tanto para la parte del seguir de línea y el cruce, como para la de la detección de las ocho señales de tráfico y la toma de decisiones ante éstas; lo cual demuestra la robustez de nuestro sistema y la eficacia con la que éste cuenta.

• Andrea González Arredondo

A lo largo de este proyecto, se usaron diferentes algoritmos, en los cuales se combinaron diferentes áreas, esto nos sirvió para que nuestro robot pudiera detectar las líneas y los cruces en los que se debía tomar una decisión, para esto usamos parte de visión computacional que trata de interpretar las imágenes que están siendo detectadas por la cámara. Para ello también se usó inteligencia artificial que fue la que nos ayudó a interpretar las señales y la luz de los semáforos, a partir de un modelo de entrenamiento.

La implementación de nuestros algoritmos con lenguaje de programación Python, nos permitió el uso de ciertas librerías para poder limpiar nuestra imagen, así como para detectarla el modelo que se usó fue Yolov5, que fue una muy buena opción ya que fue eficiente y óptima para la detección de objetos. Así como fue acoplarnos al hardware que nos habían dado, también es un área de mejora, porque existen mejores versiones para la Jetson, en los que se podría correr CUDA en el GPU y esto permitiría minimizar la latencia, de igual forma con las versiones que tuvimos que buscar para que toda nuestra implementación funcionara.

El proyecto del Puzzlebot tiene muchas áreas de mejoría, principalmente creo que debería de tener un mejor control, más desarrollado pero para esto necesitamos una mejora sensorial, ya que otros sensores nos podrían proporcionar una información más certera, con la que se nos posibilitaría que el Puzzlebot pueda seguir mejor el camino. Nuestro proyecto al final funcionó de la mejor manera posible, siempre se buscó sacar el máximo rendimiento del software y hardware, por lo que dió mucho más de lo que se esperaba.

• Aranza Leal Aguirre

En conclusión, nuestro proyecto ha sido exitoso en la implementación de algoritmos que permiten una interacción inteligente entre nuestro robot y el entorno. Los algoritmos de detección de señales y luces de tráfico basados en el modelo YOLO v5 han demostrado ser efectivos en la identificación y localización precisa de las señales en

tiempo real. Esto ha dotado al robot de capacidades de percepción visual confiables, lo que resulta fundamental para su navegación autónoma y toma de decisiones en el entorno.

Sin embargo, durante el desarrollo del proyecto también hemos identificado áreas de mejora. Uno de los principales desafíos ha sido la necesidad de ajustar y calibrar adecuadamente los sensores y actuadores del robot para garantizar su correcto funcionamiento y precisión en la interacción con el entorno. Además, hemos observado que el rendimiento de los algoritmos de detección puede verse afectado por condiciones ambientales adversas, como la iluminación o la presencia de objetos obstruyendo las señales. Por lo tanto, se requiere continuar investigando y refinando los algoritmos para lograr una mayor robustez y adaptabilidad en diferentes condiciones.

En cuanto a las áreas de mejora y trabajo futuro, consideramos que es importante explorar técnicas de fusión sensorial para combinar la información de múltiples sensores, como cámaras y sensores de proximidad, a fin de mejorar la precisión y confiabilidad de la detección y la toma de decisiones. Además, se puede considerar el uso de algoritmos de aprendizaje por refuerzo para permitir que el robot aprenda y mejore su comportamiento a medida que interactúa con el entorno.

ANEXOS

Unidad compartida en Google Drive:

https://drive.google.com/drive/folders/16Rq0hDzGM7bjF72WhfK_dAVNSuSoCscT?usp=sharing

En esta unidad compartida se puede descargar el video final de la presentación ante Manchester Robotics y NVIDIA, del cual resultamos entre los 4 mejores equipos de todos los campus a nivel nacional. Asimismo, se puede descargar el PowerPoint de la presentación utilizada.

Link al GitHub con todo el código fuente y un archivo README.md con las instrucciones para compilar el código:

https://github.com/Estebe37/ROS/tree/master/src/final_challenge

AGRADECIMIENTOS

Nos complace expresar nuestro más profundo agradecimiento a nuestro valioso socio, Manchester Robotics, por su inquebrantable dedicación y valioso apoyo a lo largo de nuestro recorrido. Valoramos enormemente las innumerables horas invertidas en nuestras clases y la generosidad con la que compartieron su vasto conocimiento y experiencia.

En particular, deseamos extender un agradecimiento especial a los distinguidos profesores Jesús Arturo Escobedo Cabello, Josué González García, José Antonio Cantoral Ceballos, Francisco Javier Navarro Barrón y Elsa Ontiveros de la UF TE3002B. Implementación de Robótica Inteligente. Sus aportes trascendentales han dejado una huella importante en nuestro desarrollo y comprensión del reto final.

REFERENCIAS

- [1] Puerto, K. (2016). Roborace es una competición con coches autónomos y eléctricos, ya están dando sus primeras vueltas sin . Xataka. <https://www.xataka.com/vehiculos/robورace-es-una-competicion-con-coches-autonomos-y-electricos-ya-estan-dando-sus-primeras-vueltas-sin-piloto>
- [2] Puerto, K. (2016). Roborace es una competición con coches autónomos y eléctricos, ya están dando sus primeras vueltas sin . Xataka. <https://www.xataka.com/vehiculos/robورace-es-una-competicion-con-coches-autonomos-y-electricos-ya-estan-dando-sus-primeras-vueltas-sin-piloto>
- [3] Piloto automático. (n.d.). Tesla México. https://www.tesla.com/es_MX/autopilot
- [4] McKeon, C. (2018). *What Is The Roborace?* V-Hr.com. <https://blog.v-hr.com/blog/what-is-the-robورace>
- [5] Taşkin Dirsehan, & Can, C. (2020). Examination of trust and sustainability concerns in autonomous vehicle adoption. 63, 101361–101361. <https://doi.org/10.1016/j.techsoc.2020.101361>
- [6] Kit de Desarrollo Jetson Nano IA - NVIDIA | MCI Electronics.cl. (2023, June 17). MCI Electronics. <https://mcielectronics.cl/shop/product/kit-de-desarrollo-jets-on-nano-nvidia-nvidia-27557/>
- [7] Roborace. (2023). Roborace: The Global Championship for Autonomous Racing. Recuperado de <https://robورace.com/>
- [8] Smith, J. (2023). Challenges and Limitations of Roborace. Recuperado de <https://robورace.com/>