Carro Seguidor de Línea Implementado con Redes Neuronales Basadas en Aprendizaje con Refuerzo

Elvis Salazar Mendez
20202005022
Julian Santiago Moreno Cucaita
20201005114
Cristian David Barragán
20211005074
Universidad Distrital Francisco José De Caldas
Facultad de Ingeniería
Proyecto Curricular Ingeniería Electrónica
Bogotá D.C. - Colombia

***Abstract*—Un carro seguidor de línea es un vehículo robótico diseñado para detectar y seguir una línea trazada en el suelo, generalmente de color negro sobre una superficie más clara. El objetivo de este proyecto es diseñar un carro seguidor de línea que complete una pista compuesta por una línea negra sobre una superficie blanca en el menor tiempo posible. Para lograrlo, se dio uso a la cámara OV7670 para aumentar el rango de visión, se implementaron redes neuronales basadas en aprendizaje por refuerzo para optimizar el desempeño y la toma de decisiones.**

***Index Terms*—Carro seguidor de línea, redes neuronales, aprendizaje por refuerzo, OV7670, Raspberry Pi Pico W**

I. Introducción

El avance de la inteligencia artificial y la robótica ha permitido el desarrollo de sistemas autónomos cada vez más sofisticados y eficientes. Un ejemplo representativo de estos sistemas es el carro seguidor de línea, un robot diseñado para seguir un camino predefinido marcado por una línea en el suelo. Tradicionalmente, estos robots utilizan sensores infrarrojos y algoritmos de control sencillos para mantener su trayectoria. Sin embargo, la integración de una cámara para ampliar el rango de visión y el uso de redes neuronales en el control de estos robots abre nuevas posibilidades, ofreciendo mejoras significativas en términos de precisión, adaptabilidad y eficiencia.

Este proyecto se centra en el diseño y desarrollo de un carro seguidor de línea controlado por redes neuronales, un enfoque innovador que combina hardware y software. La idea es aprovechar la capacidad de las redes neuronales para aprender y adaptarse a diferentes entornos y condiciones, permitiendo asi que el carro siga un camino trazado con mayor exactitud y responda de manera más eficiente a variaciones en el camino.

II. Marco Teórico

A. Raspberry Pi Pico W

La Raspberry Pi Pico W es una tarjeta de desarrollo basada en el microcontrolador RP2040, diseñada para ser una plataforma de desarrollo flexible y de bajo costo. Incluye una memoria flash de 2 MB, un cristal oscilador, un chip Infineon CYW43439 que implementa la interfaz inalámbrica WiFi de 2.4 GHz y un puerto USB. La antena está integrada en la PCB de la Pico W y cuenta con licencia de ABRACON. La interfaz inalámbrica se conecta al RP2040 a través de SPI.

Figure 1. Raspberry Pi Pico W

B. Cámara OV7670

La cámara OV7670 es un sensor de imagen CMOS de bajo voltaje que proporciona la funcionalidad completa de una cámara VGA de un solo chip y un procesador de imagen en un paquete compacto. Proporciona imágenes de 8 bits en una amplia gama de formatos controladas a través de la interfaz Serial Camera Control Bus (SCCB),

compatible con el protocolo I2C. La OV7670 puede funcionar hasta 30 fotogramas por segundo (fps) en VGA.

C. Pantalla OLED 128x32

La pantalla OLED (Organic Light-Emitting Diode) es una tecnología que emite luz propia, compuesta por una película multicapa orgánica delgada colocada entre un ánodo y un cátodo. A diferencia de la tecnología LCD, OLED no requiere una luz de fondo. Esta pantalla cuenta con una interfaz I2C que facilita su control en proyectos.

Figure 2. Pantalla OLED 128x32

D. Protocolo UART

El protocolo UART (Universal Asynchronous Receiver-Transmitter) es un método de comunicación serial asíncrona utilizado para transmitir y recibir datos entre dispositivos electrónicos. UART transmite datos bit a bit a través de líneas de comunicación dedicadas, típicamente una para transmitir (TX) y otra para recibir (RX). La comunicación se realiza sin una señal de reloj compartida, sincronizándose mediante parámetros comunes como la velocidad de baudios (baud rate).

E. Protocolo I2C

El protocolo I2C (Inter-Integrated Circuit) es un método de comunicación serial síncrona que permite la conexión de múltiples dispositivos electrónicos a través de dos líneas: una de datos (SDA) y una de reloj (SCL). Diseñado para la comunicación entre chips en cortas distancias, I2C facilita la transmisión de datos entre un maestro (controlador principal) y múltiples esclavos (dispositivos periféricos) en un bus compartido.

F. MicroPython

MicroPython es una implementación del lenguaje de programación Python 3 escrita en C, optimizada para ejecutarse en un microcontrolador. MicroPython incluye un compilador completo del lenguaje Python a bytecode y un motor e intérprete en tiempo de ejecución del bytecode que funciona en el hardware del microcontrolador. Al usuario se le presenta una línea de órdenes interactiva (REPL) que soporta la ejecución inmediata de órdenes, además de una selección de bibliotecas fundamentales de Python.

G. CircuitPython

CircuitPython es un derivado de código abierto del lenguaje de programación MicroPython dirigido a estudiantes y principiantes. Desarrollado por Adafruit Industries, es una implementación de software del lenguaje Python 3 escrita en C, adaptado para ejecutarse en varios microcontroladores modernos.

III. Procedimiento

En esta sección se detallan los planteamientos y variables considerados durante el diseño del carro seguidor de línea, haciendo énfasis en los inconvenientes presentados durante el desarrollo del proyecto.

A. Modo de Entrenamiento de la Red Neuronal

Inicialmente, se planteó que el entrenamiento de la red neuronal se llevara a cabo a través de una página web, donde las instrucciones se enviarían mediante esta plataforma. Se desarrolló un código específico para conectar la Raspberry Pi Pico W a un host externo que serviría de intermediario entre la Raspberry Pi y el dispositivo de control. Sin embargo, se observó que la velocidad de reacción del sistema era baja debido a la comunicación indirecta a través del host externo, impactando negativamente en el rendimiento del carro. Para mitigar este problema, se decidió implementar el host directamente dentro de la Raspberry Pi, estableciendo una comunicación más directa y rápida entre el emisor y el receptor.

B. Datos de la cámara

El enfoque de entrenamiento a través de la página web resultó deficiente, ya que aunque se aumentó la velocidad de reacción, no fue lo suficientemente preciso

para recopilar datos de manera eficiente que entrenaran adecuadamente la red neuronal. A partir del modelo de CircuitPython proporcionado por el docente.

Para tener un mayor control de la matriz resultante de la cámara, se estableció una condición: se definió un rango de intensidad lumínica en el que se consideraría como valor 1 aquellos valores de la cámara con baja intensidad, y como valor 0 aquellos con alta intensidad. Con este rango, se reduce la probabilidad de error ocasionado por las condiciones lumínicas en las que se realice la prueba, tomando los extremos, ya que así se asegura que se seguirá la línea negra, sin importar si hay tonalidades más claras que puedan afectar su funcionamiento. De la misma forma, este método permite un traspaso de datos mucho más rápido mediante UART, ya que de esta manera se envían solo 40 datos por renglón en lugar de enviar un número decimal."

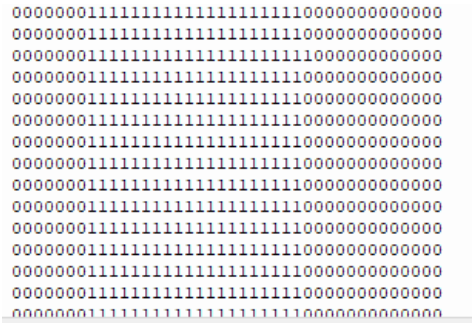

Figure 3. Matriz camara

Utilizando la misma lógica de 0 y 1, se configuró la pantalla OLED para mostrar en tiempo real lo que iba capturando el módulo de la cámara. Las dimensiones de la matriz capturada fueron de 40x30, mientras que la pantalla OLED fue de 120x32. Para aprovechar completamente la capacidad de la OLED, se multiplicó el valor del vector de 40 datos por 3, haciendo que cada dato de la cámara se repitiera tres veces, llenando completamente el display.

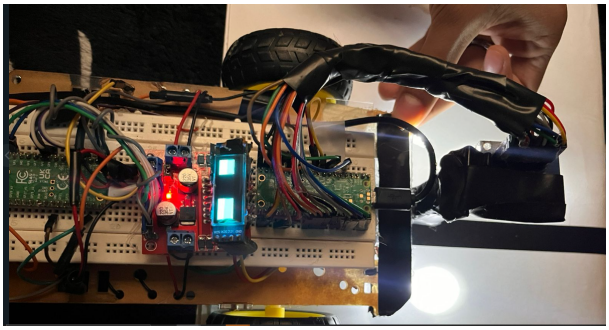

Figure 4. Visualización pista OLED.

C. Procesamiento de Datos y Decisiones

Debido al alto procesamiento que se requiere para una toma de decisión, se optó por tomar la última línea de la matriz captada por la cámara, la cual se envía mediante una conexión que utiliza el protocolo UART a 115200 baudios (velocidad máxima de transmisión UART para la Raspberry Pi Pico W), ya que a mayor velocidad las reacciones son más precisas, estableciendo una comunicación directa entre las dos Raspberry Pi. Los datos se codifican y decodifican bajo el estándar UTF-8.

Retomando el hecho de que la estrategia de la página web no dio resultado, se decidió implementar un código condicional que tomara los datos recibidos mediante el UART y que con la matriz de 40x1 se dividiera en tres secciones: forward, right y left. La dirección que tomará el carro se determina según la cantidad de unos (1) presentes en cada sección. Si la mayoría de los unos (1) se encuentran en la sección central (forward), el carro seguirá en dirección recta; si predominan en la derecha (right) o izquierda (left), el carro girará en la dirección correspondiente. Este sería el método de entrenamiento.

```
def recibir_datos_uart():
    ultima_lista_valida = None  # Variable para mantener la última lista con al menos un '1'

    while True:
        try:
            # Leer datos UART
            data_received = uart.read(1000)  # Lee una cadena de hasta 1000 bytes

            if data_received:
                try:
                    data_list = data_received.decode().strip()  # Eliminar espacios en blanco

                except UnicodeError:
                    continue
                direction = "forward"
                count_left = 0
                count_front = 0
                count_right = 0

                # Contar los unos en cada sección
                for i, value in enumerate(data_list[-40:]):
                    if value == '1':
                        if i < 10:
                            count_left += 1
                        elif 10 <= i < 30:
                            count_front += 1
                        else:
                            count_right += 1

                #print(f"Left count: {count_left}, Front count: {count_front}, Right count: {count_right}")

                    # Si hay al menos un '1' en la lista, actualizar la última lista válida
                    if '1' in data_list:
                        ultima_lista_valida = data_list

                    # Determinar la dirección en función del conteo de unos
                    if count_front > count_left and count_front > count_right:
                        direction = "forward"
                    elif count_left > count_front and count_left > count_right:
                        direction = "left"
                    elif count_right > count_left and count_right > count_front:
                        direction = "right"
                    elif count_left == 0 and count_front == 0 and count_right == 0:
                        #print(ultima_lista_valida)
                        if ultima_lista_valida:
                            for i, value in enumerate(ultima_lista_valida[-40:]):
                                if value == '1':
                                    if i < 20:
                                        direction = "left"
                                    else:
                                        direction = "right"
                                    break

                    if direction in ["left", "right"]:
                        control_motor(direction)
                        time.sleep(0.15)
                        control_motor("stop")
                    else:
                        control_motor(direction)



        except KeyboardInterrupt:
            break
recibir_datos_uart()
```

Fig. No 5 Código realizado para la comunicación entre las dos rapsberrys por medio del protocolo UART.

D. Entrenamiento de la Red Neuronal

Con el modelo de entrenamiento establecido, es momento de usar la red neuronal. Haciendo uso de la clase matriz proporcionada por el docente, se fijaron los métodos para inicializar el perceptrón (modelo básico de red neuronal), proporcionando el número de datos a ingresar y el número de salidas esperadas, tanto para el entrenamiento como para la predicción. En el caso de este proyecto, son los 40 datos de la última línea de la matriz y las tres direcciones planteadas anteriormente como salidas, usadas inicialmente para calcular aleatoriamente los pesos y sesgos. Basado en lo anterior y con ayuda de los métodos "Train" y "save-file", se guarda en un documento de texto el entrenamiento correspondiente, el cual incluye la cantidad de datos de entrada, salida, pesos y sesgos con

una magnitud de 40x3, calculados para cada una de las salidas mediante el entrenamiento.

E. Inicialización de la Red Neuronal

Inicializando el perceptrón con los datos entrenados, mediante el método de la función PERCEPTRON.Loadfile, se carga el valor actual o el mejor entrenamiento de pesos y sesgos para cada una de las salidas. Entonces, posteriormente a recibir los datos del UART, se convierte dicha lista resultante en una matriz de 40x1 tomada como entrada (input) del perceptrón para el método ”Predict”, el cual calcula la probabilidad de cada una de las salidas. Es decir, retorna una matriz de 3x1 en la que el mayor valor es la dirección que deben tomar los motores. Bajo condicionales y tomando un diccionario de Direcciones, se establece el curso que deben tomar los motores bajo 3 matrices de 3x1 cada una diseñada para una dirección en específico, enviando el resultado a la función ”control-motor”, encargada de encender los motores bajo la instrucción enviada.

F. Entrenamiento por Refuerzo

El entrenamiento por refuerzo también parte de la Clase matriz, inicializa de forma similar al perceptrón, añadiendo una función de error requerida para ir ajustando el funcionamiento del perceptrón, que en el caso de este proyecto se optó por el error cuadrático medio. También solicita una lista de las salidas en términos matriciales. El aprendizaje por refuerzo también hace uso de los datos entrantes, inicializando dos perceptrones: uno de control y otro de modelo futuro. El de control, como su nombre lo indica, controla la dirección actual del carro, mientras que el de modelo futuro predice la próxima dirección. Al comparar bajo la función de error, busca ir mejorando consecutivamente la toma de decisiones. Posterior a la decodificación de los datos del UART y a la conversión matricial, se llama al método ”predict” de la función ”reinforcement”, y bajo los mismos criterios de probabilidad del manejo por un solo perceptrón, se establece la dirección para luego llamar a la función ”control-motor”.

```
def control_motor(direction):
    # Envía la dirección por UART

    if direction == "forward":
        in1_motor1.value(0)
        in2_motor1.value(1)
        in1_motor2.value(0)
        in2_motor2.value(1)
    elif direction == "backward":
        in1_motor1.value(1)
        in2_motor1.value(0)
        in1_motor2.value(1)
        in2_motor2.value(0)
    elif direction == "left":
        in1_motor1.value(0)
        in2_motor1.value(1)
        in1_motor2.value(0)
        in2_motor2.value(1)
    elif direction == "right":
        in1_motor1.value(0)
        in2_motor1.value(1)
        in1_motor2.value(0)
        in2_motor2.value(1)
    else:
        # Detener ambos motores si no se especifica una dirección vál
        in1_motor1.value(0)
        in2_motor1.value(0)
        in1_motor2.value(0)
        in2_motor2.value(0)
# Ajustar la velocidad según la dirección
    if direction == "left":
        # Configuración para girar a la izquierda
        pwm_motor1.freq(2000)  # Frecuencia PWM de 1000 Hz para el motor 1
        pwm_motor1.duty_ns(int(0*65535)) # Reducir la velocidad del motor 1
        pwm_motor2.freq(5000)  # Frecuencia PWM de 1000 Hz para el motor 2
        pwm_motor2.duty_u16(int(0.91*65535))  # Mantener la velocidad del motor 2 constant
    elif direction == "right":
        # Configuración para girar a la derecha
        pwm_motor1.freq(5000)  # Frecuencia PWM de 1000 Hz para el motor 1
        pwm_motor1.duty_u16(int(0.91*65535))  # Mantener la velocidad del motor 1 constante
        pwm_motor2.freq(2000)  # Frecuencia PWM de 1000 Hz para el motor 2
        pwm_motor2.duty_ns(int(0*65535))  # Reducir la velocidad del motor 2
    elif direction == "backward":
        # Configuración para detener ambos motores
        pwm_motor1.freq(200)  # Frecuencia PWM de 5000 Hz para el motor 1
        pwm_motor1.duty_u16(int(0.7*65535))  # Máxima velocidad del motor 1
        pwm_motor2.freq(1000)  # Frecuencia PWM de 5000 Hz para el motor 2
        pwm_motor2.duty_u16(int(0.7*65535))  # Reducir la velocidad del motor 2
    else:
        # Configuración para otras direcciones (forward,
        pwm_motor1.freq(10000)  # Frecuencia PWM de 5000 Hz para el motor 1
        pwm_motor1.duty_u16(int(0.88*65535))  # Máxima velocidad del motor 1
        pwm_motor2.freq(10000)  # Frecuencia PWM de 5000 Hz para el motor 2
        pwm_motor2.duty_u16(int(0.83*65535))
# Mantener la velocidad del motor 2 constante  # Mantener la velocidad del motor 2 constant
```

Fig. No 4 Código realizado para el control de los motores

Figure 5. Codigo ”control-motor”.

G. Control de Velocidad del Carro

En este proyecto, no se utilizó una red neuronal para ajustar la velocidad del carro. En su lugar, se definió una velocidad fija para cada rueda en diferentes direcciones, lo cual simplificó el control del movimiento, permitiendo centrar los esfuerzos en el desarrollo del seguidor de línea utilizando la clase matriz para implementar el perceptrón.

IV. Resultados

Se evidencia una clara alteración de las decisiones dependiendo del entorno en el que se realice la prueba. Las condiciones lumínicas, como sombras y posición de la luz, son un factor clave para el buen funcionamiento del modelo. Aunque se hayan tomado medidas como la implementación de LED para mitigar el efecto de las sombras y lo mencionado en el apartado de la cámara referente al rango establecido por intensidades, no siempre resulta ser eficiente.

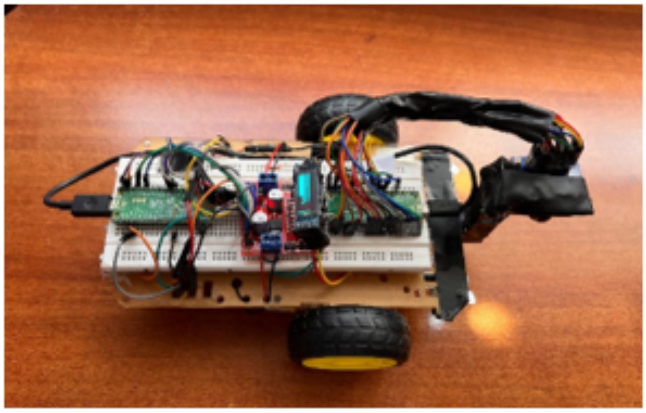

Fig. No 6 Vista periferica del carro seguidor de línea.

En lo que respecta a la toma de decisiones del modelo, el método de entrenamiento resulta ser el factor más importante, ya que un buen cálculo de pesos y sesgos hace que el desempeño mejore en cada etapa del proyecto. Se evidenció que como seguidor de línea básico sin red neuronal, cualquier alteración o campo en blanco hacía que el carro cambiara su curso, tardando (1:52 min) en completar una vuelta a la pista de prueba. Con la implementación del perceptrón se evidencia que, debido al alto procesamiento, este tardaba más en la toma de decisiones, por lo cual se tuvo que poner condiciones de stop para que tuviese tiempo de analizar, predecir y tomar la decisión, resultando así en un tiempo de (2:30 min) en dar la vuelta a la pista de prueba, que a diferencia del primer modelo era mucho más lento pero más preciso. Finalmente, con el aprendizaje por refuerzo, se llegó a un punto medio en lo que respecta a velocidad y tiempo. Se pudo disminuir las condiciones de velocidad y lograr un mejor tiempo (1:15 min) dando la vuelta a la pista.

Fig. No 8 Recorrido realizado por el carro seguidor de línea

V. Conclusiones

El proyecto logró implementar con éxito un carro seguidor de línea utilizando redes neuronales basadas en aprendizaje por refuerzo, controlado por una cámara OV7670 y una Raspberry Pi Pico W. A pesar de los desafíos técnicos encontrados, como incompatibilidades entre OLED, módulos y demás, los resultados mostraron una mejora en la precisión y adaptabilidad del carro en comparación con métodos tradicionales. El proceso a seguir de entrenamiento, perceptrón y aprendizaje es un método eficiente para distinguir el mejoramiento continuo de cada uno de los modelos, teniendo así una referencia para pulir.

VI. Referencias

- Gerardo Muñoz, *GitHub Repository*, https://github.com/GerardoMunoz
- Raspberry Pi, *Raspberry Pi Pico Documentation*, https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html