

Diseño de un Seguidor de Línea Con Red Neuronal en Microcontrolador RP2040

Juan Camilo Rodríguez García - 20202005070

Juan Sebastian Casas Barbosa - 20211005031

Haider Santiago Calderón Rodríguez - 20211005075

Universidad Distrital Francisco José de Caldas, Bogotá, Colombia

Proyecto Curricular de Ingeniería Electrónica

Diseño digital con microcontroladores

Docente: Gerardo Alcides Muñoz Quiñonez

Resumen—Este informe describe el desarrollo de un robot seguidor de línea utilizando el microcontrolador RP2040. Se emplearon dos placas Raspberry Pi Pico W que se comunican mediante UART. Una placa capturó imágenes y analizó la línea con CircuitPython, mientras que la otra controló los motores con MicroPython. Se hizo especial hincapié en la implementación de una red neuronal para el seguimiento de la línea. El robot demostró una notable capacidad para seguir la línea con eficiencia y precisión, completando el recorrido utilizando la red neuronal mencionada. A lo largo del documento se detallan las dificultades encontradas y las soluciones implementadas para superarlas.

Index Terms—RP2040; Red neuronal; seguidor de línea; Microcontrolador.

I. DESCRIPCIÓN DEL PROYECTO

En este proyecto, se desarrolló un robot seguidor de línea diseñado para seguir una pista marcada en el suelo. Se hizo uso de dos placas Raspberry Pi Pico W para dividir las tareas: una placa se encarga de capturar imágenes a través de una cámara y realizar el análisis de la línea utilizando CircuitPython, mientras que la otra placa controla los motores y la dirección del robot utilizando MicroPython.

Para la comunicación entre las placas, empleamos la técnica de transferencia de datos UART (Universal Asynchronous Receiver-Transmitter). Esto permitió una comunicación eficiente y en tiempo real entre ambas placas, asegurando una coordinación adecuada para seguir la línea de manera efectiva.

Uno de los aspectos clave del proyecto es el uso de una red neuronal, específicamente la clase Perceptrón implementada en MicroPython, para entrenar al robot y mejorar su capacidad de seguir la línea de manera más precisa. Esta red neuronal fue fundamental para ajustar los parámetros del robot en función de las condiciones del entorno y mejorar su desempeño en diferentes situaciones.

Además, para garantizar un funcionamiento óptimo, utilizamos una combinación de lenguajes de programación: CircuitPython para el análisis de imágenes y control de la cámara, y MicroPython para el control de los motores y la implementación de la red neuronal. Esta elección nos

permitió aprovechar las ventajas de cada lenguaje y optimizar el rendimiento del robot en su conjunto. Al final, se logró que el robot siguiera líneas con gran precisión, cumpliendo con el objetivo principal.

II. DISEÑO DEL PROYECTO

II-A. Requerimientos de Diseño

Para el desarrollo del robot seguidor de línea, se establecieron una serie de requerimientos. El primer requerimiento fue utilizar la placa Raspberry Pi Pico W, basada en el microcontrolador RP2040, para el control y procesamiento de datos. Esta elección asegura que el robot tenga la capacidad de manejar las tareas complejas necesarias para el seguimiento de línea.

Se indicó también implementar la cámara OV7670 para la captura de imágenes de la pista que el robot debe seguir. Esta cámara ofrece la resolución y velocidad necesarias para un análisis en tiempo real. Complementando esto, se debía incluir una pantalla OLED para mostrar información relevante sobre el estado del robot, como las velocidades de los motores y el valor del error calculado.

Otro requerimiento crucial fue la implementación de una red neuronal para la detección y seguimiento de la línea. Los datos capturados por la cámara son procesados por una Raspberry Pi Pico W, y la red neuronal interpreta estos datos para ajustar la trayectoria del robot de manera precisa y eficiente. Este enfoque permite que el robot se adapte en tiempo real a los cambios en la trayectoria de la línea.

El objetivo final establecido fue que, al finalizar el desarrollo, el robot debía ser capaz de seguir una línea marcada en el suelo con precisión y eficiencia, ajustando su trayectoria en tiempo real según la información procesada. Estos requerimientos fueron esenciales para guiar el diseño y desarrollo del proyecto, asegurando que el robot cumpliera con las expectativas de seguimiento de línea de manera efectiva y confiable.

II-B. Esquema General de Diseño

Para el diseño se partió del siguiente diagrama de bloques

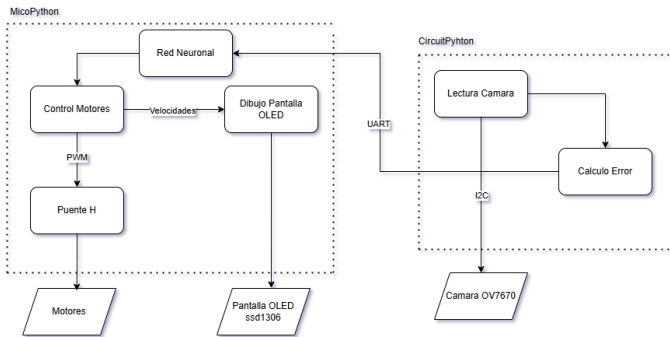


Figura 1: Diagrama de Bloques Seguidor de Línea

Haciendo énfasis en cada una de las partes de este esquema se tiene lo siguiente:

■ Cámara - Sensor

El código `code.py`, ejecutado en la primera Raspberry Pi, configura la cámara **OV7670** utilizando el bus I2C. Esta cámara captura imágenes y las convierte a una escala de grises, donde los valores de los píxeles se representan mediante caracteres `#` para negro (indica la línea) y `-` para blanco (fondo). Esta simplificación permite identificar fácilmente la línea negra que el robot debe seguir. La cámara está configurada para capturar imágenes en una resolución de 16x16 píxeles, lo que es suficiente para el seguimiento de líneas y reduce la cantidad de datos a procesar.

```
1 cam_bus = busio.I2C(board.GP21, board.GP20)
2
3 #Definimos las conexiones fisicas de la camara.
4
5 cam = OV7670(
6     cam_bus,
7     data_pins=[
8         board.GP0,
9         board.GP1,
10        board.GP2,
11        board.GP3,
12        board.GP4,
13        board.GP5,
14        board.GP6,
15        board.GP7,
16    ],
17    clock=board.GP8,
18    vsync=board.GP13,
19    href=board.GP12,
20    mclk=board.GP9,
21    shutdown=board.GP15,
22    reset=board.GP14,
23 )
```

Código 1: Configuración OV7670 I2C.

```
1 cam.size = OV7670_SIZE_DIV16
2 cam.colorsapce = OV7670_COLOR_YUV
3 cam.flip_y = True
4 buf = bytearray(2 * cam.width * cam.height)
5 chars = b"#" * 128 + b"-" * 128 #Definimos una
6   lectura de solo oscuro y claro.
7 width = cam.width
8 row = bytearray(2 * width)
```

Código 2: Inicialización Cámara.

El análisis de datos comienza dividiendo la imagen capturada en varias filas y evaluando el número de píxeles negros en cada una de ellas. El código se centra en un rango específico de filas que considera de interés para detectar la línea. La imagen se divide en dos mitades, y se calcula un error basado en la diferencia de píxeles negros entre ambas mitades. Este error se interpreta como una medida de desvío de la línea respecto al centro de la cámara: los valores negativos indican un desvío a la izquierda y los positivos, a la derecha. Este método permite determinar la posición relativa de la línea con respecto al robot.

```
1 while True:
2
3     cam.capture(buf) #LLamamos al listado de
4     datos de la camara.
5     error_suma = 0 #Suma de errores para
6     calcular el promedio al final.
7     cantidad_filas = cam.height # N mero total
8     de filas con base a la camara.
9     filas_interes = range(20, 21) # Filas de
10    interes para elavluar.
11
12    #Realizamos ahora un barrido de los elemntos
13    dentro de las finlas de interes.
14
15    for j in filas_interes:
16
17        error_filaN,error_filap = 0,0 # Error
18        para la fila actual entre positivos y
19        negativos.
20        row_index = width * j * 2
21
22        #Definimos un rango de evaluo dentro de
23        las filas asi como un salto para reducir
24        los tiempos de analisis.
25
26        for i in range(26, 54, 2):
27
28            #LLamamos la inFormacion.
29            pixel_value = buf[row_index + i]
30            char_index = pixel_value * (len(
31            chars) - 1) // 255
32            char_value = chars[char_index]
33            row[i] = row[i + 1] = char_value
```

Código 3: Definiciones y Rango Cámara.

El valor del error se transmite continuamente a la otra Raspberry Pi a través del bus UART. Este error es crucial para ajustar las velocidades de los motores del robot, permitiendo que siga la línea de manera precisa. La transmisión de datos en tiempo real asegura que el robot puede responder rápidamente a cualquier cambio en la posición de la línea, mejorando su capacidad para seguir la línea de manera constante. El flujo continuo de datos entre la cámara y la unidad de control es esencial para la eficacia del sistema.

■ Motores

El código `main.py`, ejecutado en la segunda Raspberry Pi, es responsable del control de los motores y de otras tareas auxiliares. Al inicio del código, se configuran los pines necesarios para los LEDs, el bus UART para la comunicación con la otra Raspberry Pi, y los pines PWM para controlar los motores. Los motores se ajustan en tiempo real según el valor del error recibido de la cámara,

lo que permite que el robot siga la línea de manera precisa. Los motores se configuran con una frecuencia de 500 Hz y se inicializan con una velocidad de 0.

```
1 if train==True:#De ser cierto hablamos de
    nuestro metodo para entrenamiento
    automatico.
2
3     multiplier=655.55/3#Variable que regula la
    velocidad de los motores.
4
5     v_d_Porcentual=int(min(100,(100+LINE_MATRIX
    [0,0])))#Proceso encargado de la velocidad
    porcentual del motor derecho.
6     v_i_Porcentual=int(min(100,(100-LINE_MATRIX
    [0,0])))#Proceso encargado de la velocidad
    porcentual del motor izquierdo.
7
8     v_d=int(v_d_Porcentual*multiplier)#
    Regulacion de la velocidad a valores de
    entre 65555 y 0 para motor derecho.
9     v_i=int(v_i_Porcentual*multiplier)#
    Regulacion de la velocidad a valores de
    entre 65555 y 0 para motor izquierdo.
```

Código 4: Control Motores.

El sistema incluye un modo de entrenamiento y un modo de aplicación de la red neuronal, controlados mediante el pin `gp0_pin`. En el modo de entrenamiento, los pesos de la red neuronal se ajustan en función del error y las velocidades de los motores, lo que permite al sistema aprender cómo seguir la línea más eficientemente. En el modo de aplicación, se utilizan los pesos entrenados para predecir las velocidades de los motores basándose en el error recibido. Este proceso de aprendizaje continuo mejora la capacidad del robot para seguir la línea en diversas condiciones.

Durante el modo de aplicación, la velocidad de los motores se ajusta según las predicciones de la red neuronal. La función `train_control` maneja el entrenamiento y la aplicación de la red neuronal, ajustando las velocidades de los motores y actualizando la pantalla OLED para mostrar las velocidades y el error en tiempo real. Este enfoque permite un control preciso y adaptativo del robot, asegurando que pueda seguir la línea de manera efectiva. La capacidad de alternar entre el entrenamiento y la aplicación de la red neuronal proporciona una gran flexibilidad y robustez al sistema.

■ Transmisión de Datos

La comunicación entre las dos Raspberry Pis se realiza utilizando UART (Universal Asynchronous Receiver-Transmitter). La Raspberry Pi que controla la cámara transmite continuamente el valor del error calculado a la otra Raspberry Pi, que utiliza esta información para ajustar los motores del robot. Esta transmisión es esencial para el funcionamiento en tiempo real del sistema, permitiendo que el robot responda rápidamente a los cambios en la posición de la línea. La configuración del UART en ambas Raspberry Pis asegura una comunicación fluida y constante.

```
1 uart.write(str(error_promedio).encode() + b"\r\n
    ")#Enviamos la informacion mediante la uart
    .
```

Código 5: Transmisor Circuitpython.

```
1 try:
2
3     datos_recibidos = uart.read(1) # Leer
    un byte de datos
4
5     if datos_recibidos:
6
7         char = datos_recibidos.decode('utf-8
        ', 'replace')
8
9         if char == '\r': # Si se recibe
            retorno de camara
10
11            try:
12
13                valor = int(received_text.
                strip()) #Transformamos el valor recibido
                a un numero entero.
```

Código 6: Receptor Micropython.

■ Red Neuronal

La red neuronal implementada en el código `main.py` es un perceptrón simple, diseñado para predecir las velocidades de los motores basándose en el error de la línea. Durante el entrenamiento, el perceptrón ajusta sus pesos utilizando la diferencia entre la salida deseada (velocidades de los motores) y la salida actual. Este proceso permite que el sistema aprenda a ajustar las velocidades de los motores de manera más precisa, mejorando su capacidad para seguir la línea detectada. La clase `Perceptron` maneja la creación, entrenamiento y predicción utilizando matrices para representar los datos.

```
1 class Perceptron:
2
3     #Definimos las caractereisticas de la matriz
    de pesos con esta definicion.
4
5     def __init__(self, ENTRADAS, SALIDAS=None):
6
7         #Realizamos una condicion que nos diga
        si existe informacion en la matriz de pesos
        o si no generar una aleatoria.
8
9         if isinstance(ENTRADAS, str):
10
11             self.weights = Matrix.load_file(
                ENTRADAS)
12
13         else:
14
15             self.weights = Matrix(ENTRADAS+1,
                SALIDAS+1, [random.random() for _ in range
                ((ENTRADAS+1) * (SALIDAS+1))])
16
17     #Definimos la manera en la que trabajaremos
    las predicciones de las variables de salida
    con la siguiente definicion.
18
19     def predict(self, inputs):
20
21         tail=False
22
23         if inputs.n==self.weights.m-1:
24
25             result = Matrix.untail(Matrix.tail(
                inputs) * self.weights)
```

```

26         else:
27
28             result = inputs * self.weights
29
30         return result
31
32     #Definimos la forma en la cual traajamos el
33     #entrenamiento de la red con base en datos
34     #introducidos a esta definicion.
35
36     def train(self, inputs, labels,
37               learning_rate=0.01, epochs=1):
38
39         if inputs.n==self.weights.m-1:
40
41             inputs=Matrix.tail(inputs)
42
43         if labels.n==self.weights.n-1:
44
45             labels=Matrix.tail(labels)
46
47         for epoch in range(epochs):
48
49             predictions = self.predict(inputs)
50
51             error = labels - predictions
52
53             self.weights=self.weights.add_tail(
54             inputs.T() * error * learning_rate)
55
56     #Definimos la manera en la cual aseguraremos
57     #que se guarde la matriz de pesos.
58
59     def save_file(self, name):
60
61         self.weights.save_file(name)

```

Código 7: Clase Perceptron.

El proceso de entrenamiento se lleva a cabo cuando el sistema está en modo de entrenamiento, controlado por el pin `gp0_pin`. En este modo, los valores de error recibidos se utilizan para ajustar los pesos de la red neuronal. El código utiliza un algoritmo de retropropagación para ajustar los pesos, minimizando el error entre las salidas previstas y las reales. Este proceso de ajuste continuo permite que la red neuronal mejore su precisión con el tiempo, adaptándose mejor a las condiciones de seguimiento de la línea.

En el modo de aplicación, la red neuronal utiliza los pesos entrenados para predecir las velocidades de los motores basándose en el error recibido. Los pesos se pueden guardar en un archivo (`carro.txt`) y cargar desde él para reutilizarlos en futuras sesiones, lo que permite mantener el aprendizaje del sistema entre ejecuciones. La capacidad de aprender y adaptarse es crucial para el funcionamiento eficiente del robot en diversas condiciones de seguimiento de la línea. Esta implementación de una red neuronal simple proporciona un control adaptativo que mejora significativamente la capacidad del robot para seguir la línea de manera precisa y confiable.

■ Periféricos

En nuestro caso con periférico hacemos referencia a la pantalla OLED que se configura al inicio del código

`main.py` utilizando el bus I2C, permitiendo una interfaz gráfica para mostrar información relevante sobre el estado del robot. La función `draw_tachometer` es responsable de actualizar la pantalla, mostrando las velocidades de los motores izquierdo y derecho, así como el valor del error calculado.

La pantalla OLED se actualiza en cada ciclo de control de motores, reflejando cualquier cambio en las velocidades o en el error detectado. Esta actualización constante asegura que el operador del robot pueda ver en tiempo real cómo el sistema está respondiendo a la línea que sigue.

III. MONTAJE

III-A. Esquema de conexiones

RASPBERRY (CIRCUITPYTHON)	
CAMERA OV7670	
D0	GP0
D1	GP1
D2	GP2
D3	GP3
D4	GP4
D5	GP5
D6	GP6
D7	GP7
PLK	GP8
XLK	GP9
HS	GP12
VS	GP13
RE	GP14
PWDN	GP15
SDA	GP20
SCL	GP21
3,3 V	3,3 V_OUT
DGND	GND
UART	
TX	GP16
RX	GP17
OLED	
VDD	3,3 V_OUT
GND	GND
EXPLORING LIGHT	
CATHODE	GND
BATTERY (3,7 V)	
POSITIVE	VSYS
NEGATIVE	GND

RASPBERRY (MICROPYTHON)	
TRAIN LEDs	
RED	GP18
GREEN	GP19
OLED	
SDA	GP26
SCK	GP27
UART	
TX	GP4
RX	GP5
TRAIN CONTROL	
GP0	3,3 V_OUT / GND
HW-095	
IN2	GP16 (RIGTH)
IN3	GP17 (LEFT)
EXPLORING LIGHT	
ANODE	3,3 V_OUT
BATTERY (3,7 V)	
POSITIVE	12 V (HW-095)
NEGATIVE	GND (HW-095)

III-B. Montaje Físico

A partir del esquema de conexiones que parte de los códigos de la cámara y los motores se realizó el montaje físico para el cual las conexiones entre las Picos y los periféricos se hicieron con cable UTP utilizando como base final para el chasis del seguidor, baqueas sobre las cuales se podía dar forma al carro y a la vez incluir sobre el chasis switches para el apagado y prendido del puente H o las raspberries o así mismo los módulos de carga de las baterías, los materiales utilizados para la realización del montaje físico se muestran en los anexos, a continuación se mostrara una serie de pasos generales sobre la construcción y se mostraran las fotos de los avances en la forma y construcción del carro.

1. Conexión Raspberry con CircuitPython a Cámara 0V7670
2. Conexión física UART entre Raspberrys
3. Conexión del modulo Puente H a la Raspberry con MicroPython
4. Creación del Chasis del carro con baquelas y tornillos
5. Montaje de las raspberrys, el Puente H y la cámara sobre el chasis
6. Adición de los moto-reductores al chasis y conexión con el modulo Puente H
7. Adición de switches para prendido y apagado de las raspberrys y el Puente H
8. Adición de la batería que alimenta el carro (Se recomendando que sean 2, una para el Puente H (9v) y otra para las Raspberrys (3.7v) ambas recargables)
9. Inclusión del modulo de carga para la batería de 3.7V
10. Ajustes Finales de posición y enfoque de la cámara o posición de las ruedas en el chasis

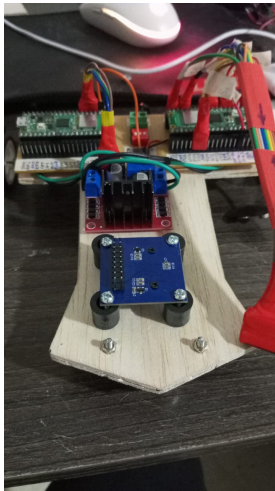


Figura 2: Modelo 1 Chasis Madera

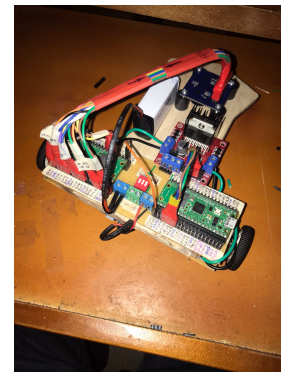


Figura 3: Modelo 2 Pines Batería

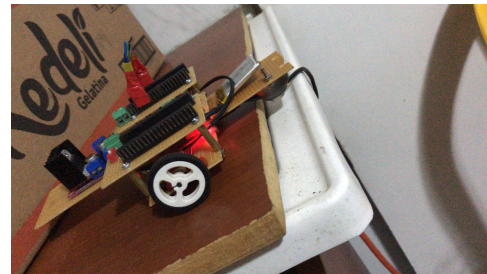


Figura 4: Modelo 3 Chasis Baquela



Figura 5: Modelo 4 Chasis Baquela 2

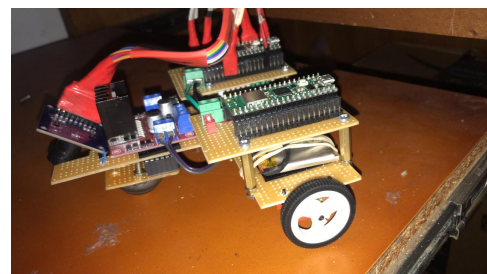


Figura 6: Modelo 5

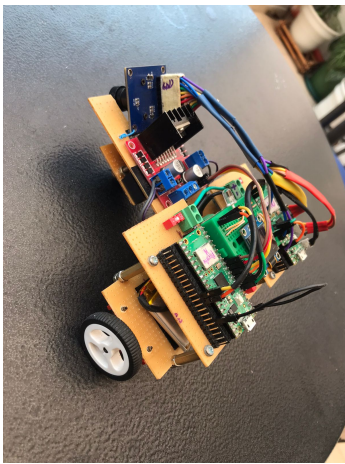


Figura 7: Modelo 6 Chasis Baquela Final y Ajuste Posicion de Camara

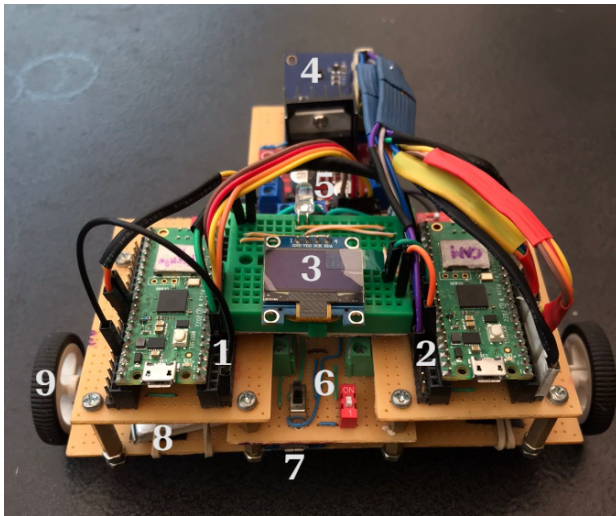


Figura 8: Modelo Final

Para el modelo final se tiene según la numeración:

1. Raspberry Motores MicroPython
2. Raspberry Camara CircuitPython
3. Pantalla Oled ssd1306
4. Camara OV7670
5. Puente H
6. Switches Encendido y Apagado Puente H y Microcontroladores
7. Modulo de carga bateria 3.7v
8. Baterías tanto a la izquierda (3.7v) como a la derecha (9v)
9. Llantas y Moto-reductores

IV. DIFICULTADES Y MEJORAS REALIZADAS

A lo largo de el diseño y construcción del Seguidor de linease tuvieron una serie de dificultades que poco a poco se tuvieron que ir resolviendo hasta llegar a un diseño o versión mas óptima que la anterior, esto se muestra a continuación:

IV-A. Por Código

■ Velocidad de Lectura de Datos

El principal inconveniente con el que nos enfrentamos fue la velocidad de lectura de datos, esto debido principalmente al modo en el que se procesaban los datos de la cámara ya que en un inicio se procesaban todas las filas de la imagen (de la 10 a la 21), lo cual generaba que cada ciclo de la cámara tardara un tiempo considerable que afectaba al funcionamiento del carro. Por este motivo, para la entrega final, solo se analiza una fila de interés específica (la fila 20). Esto reduce la cantidad de datos procesados en cada ciclo, acelerando la tasa de actualización de los datos de la cámara.

■ Lógica de Calculo de Error

El cálculo del error también ha sido mejorado para la entrega final. Anteriormente, se calculaba un error promedio basado en todos los píxeles de cada fila de interés, lo cual podía ser ineficiente y menos preciso. En el código final, el error se calcula de manera más precisa y directa, comparando la suma de los errores en dos secciones de la fila: negativa y positiva. Esta nueva lógica permite una detección más rápida y precisa de la desviación de la línea, ya que considera las diferencias de intensidad en segmentos específicos, facilitando un cálculo de error más eficiente y exacto.

■ Transmisión de Datos

Para la entrega final se mejoró la decodificación y manejo de los datos recibidos, implementando validaciones por medio de condicionales 'if' para asegurar que solo se procesen datos válidos. Además, se agregó un control de estado que determina si se está en modo de entrenamiento o de aplicación, ajustando así el comportamiento de la transmisión y recepción de datos, mejorando el rendimiento de la comunicación entre las Raspberry Pi.

IV-B. Físicas

■ Iluminación

Al probar el carro, observamos que su comportamiento variaba considerablemente según las condiciones de iluminación del entorno. Para garantizar resultados consistentes, decidimos incorporar un conjunto de LEDs en serie a la estructura del carrito. Esta adición asegura que la cámara siempre reciba la misma iluminación, independientemente de la luz ambiental, lo que estandariza el comportamiento del vehículo durante las pruebas.

■ Estructura del carro y Posición de la Cámara

La estructura del carro se adaptó continuamente a medida que avanzábamos en el proyecto y añadíamos nuevas funcionalidades, como los LEDs de iluminación y la pantalla OLED. Además, a través de pruebas iterativas, reubicamos la cámara para que quedara posicionada con un ángulo específico. Esta nueva posición permitió que la cámara anticipara la información de la línea negra a

seguir. Gracias a esto, al utilizar la red neuronal, las predicciones se realizaban con suficiente anticipación para que el carro pudiera girar y tomar decisiones de forma rápida y eficiente.

V. RESULTADOS OBTENIDOS

El proyecto concluyó con resultados satisfactorios, pues durante las pruebas finales, el robot demostró una capacidad destacable para seguir una pista marcada en el suelo con precisión y eficiencia. Las mejoras implementadas en el procesamiento de datos y en la estructura física del carro contribuyeron significativamente a estos resultados positivos.

El tiempo de respuesta del robot fue notablemente rápido. En el circuito de prueba, el robot logró completar el recorrido en menos de 20 segundos, manteniéndose firmemente sobre la línea en todo momento. Este rendimiento demuestra no solo la efectividad de la red neuronal implementada para la detección y seguimiento de la línea, sino también la integración exitosa de los componentes de hardware y software que conforman el sistema del robot.

VI. ANALISIS DE RESULTADOS Y CONCLUSIONES

Las mejoras realizadas en el procesamiento de datos ha sido crucial para optimizar el rendimiento del sistema. Las mejoras en la velocidad de lectura de datos y en la lógica de cálculo del error han contribuido significativamente a una respuesta más rápida y precisa ante desviaciones en la pista. Este aspecto ha sido fundamental para garantizar un seguimiento suave y continuo, minimizando el tiempo de reacción del robot y mejorando su capacidad de ajuste.


La integración de componentes físicos y lógicos ha sido otro punto destacado. La incorporación estratégica de LEDs para mantener una iluminación constante y la reubicación óptima de la cámara han sido decisiones clave para estandarizar el comportamiento del robot en diferentes condiciones de luz, garantizando así una detección confiable y consistente de la línea.









VII. ENLACE CÓDIGO GITHUB

En el siguiente link en la carpeta **Seguidor de línea** se encuentra el enlace a los códigos de ambas raspberries, circuit python y micropython de los cuales fueron tomados los pequeños pedazos de código que se mostraron en este informe para el entendimiento del desarrollo del proyecto <https://github.com/SebastianXV2004/Juan-Sebastian-Casas-Barbosa.git>

VIII. ANEXOS

DISPOSITIVO	NOMBRE	CANTIDAD	PRECIO UNITARIO	TOTAL
	Raspberry Pi Pico W	2	\$ 50.000,00	\$ 100.000,00
	HW-095	1	\$ 12.000,00	\$ 12.000,00
	TP4056	1	\$ 3.900,00	\$ 3.900,00
	OV7670	1	\$ 20.900,00	\$ 20.900,00
	Display LCD OLED 128x64 1.3"	1	\$ 29.000,00	\$ 29.000,00
	Motoreductor N20 1000rpm 1k	2	\$ 28.900,00	\$ 57.800,00
	BATERIA RECARGABLE LITIO 3.7V - 500mA	1	\$ 25.900,00	\$ 25.900,00
	Pila 9v Recargable Con Cargador	2	\$ 45.000,00	\$ 90.000,00
	Rueda Loca Metalica Con Esfera	1	\$ 7.600,00	\$ 7.600,00

	Llanta Pololu 34x7mm N20	2	\$ 5.500,00	\$ 11.000,00
	Soporte Plástico para Motorreductor N20	2	\$ 1.700,00	\$ 3.400,00
	Dipswitch 1 Posicion	3	\$ 700,00	\$ 2.100,00
	Bornera de 2 Pines con Tornillos	5	\$ 500,00	\$ 2.500,00
	SEPARADOR HEXAGONAL 30 -3cm	4	\$ 800,00	\$ 3.200,00
	SEPARADOR HEXAGONAL 10 -1.5cm	8	\$ 600,00	\$ 4.800,00
	Regleta de pines Header Macho Macho 1x40	2	\$ 800,00	\$ 1.600,00
	Regleta de pines en L Header Macho Macho 1x40 Angulo 90°	1	\$ 1.000,00	\$ 1.000,00
	Regleta de pines Header Macho Hembra 2x40	3	\$ 1.900,00	\$ 5.700,00

	LEDS 5mm COLORES SURTIDOS	5	\$ 150,00	\$ 750,00
	SWITCH CORREDERA PEQUEÑO	1	\$ 600,00	\$ 600,00
	Jumpers Dupont 30cm Hembra Hembra X10	2	\$ 2.900,00	\$ 5.800,00
	Jumpers Dupont 20cm Macho Macho X10	1	\$ 2.000,00	\$ 2.000,00
	Resistencias	2	\$ 50,00	\$ 100,00
	TRIMER 1K OHM REFERENCIA 102	1	\$ 900,00	\$ 900,00
	Baquela Universal tipo Protoboard	4	\$ 3.900,00	\$ 15.600,00
	SOLDADURA DE ESTAÑO 60/40 1mm (metro)	3	\$ 1.900,00	\$ 5.700,00
TOTAL				\$ 413.850,00