Carrito Seguidor De Línea Con Aprendizaje Por Refuerzo

Giussepe Darío Sanabria Combariza – 20142005027
Yeimy Camila Morales Bedoya – 20191005132
Luis Miguel Arevalo Becerra 20192005098
Universidad Distrita Francisco Jose de Caldasl
Ingenieria Electronica
Bogota, Colombia

***Resumen*—**

This paper presents the development of an autonomous line-following cart using Raspberry Pi Pico microcontrollers and neural networks with reinforcement learning. The system configuration includes an OV7670 camera for capturing images of the black line, which are processed in real-time to guide the cart. An integrated H-bridge module controls the motors that drive the wheels, allowing precise adjustments in direction and speed. The implementation of reinforcement learning optimizes the cart's behavior, enhancing its ability to accurately and efficiently follow the line. The results demonstrate the effectiveness of the proposed approach in automating line-following tasks, highlighting the potential of computer vision and machine learning technologies in mobile robotics applications.

***Keywords*—** red neuronal, microcontrolador, seguidor línea

I. Introducción

En la actualidad, la automatización y la inteligencia artificial están transformando múltiples sectores, desde la manufactura hasta el transporte. Un área específica de interés es la robótica móvil, donde los sistemas autónomos son capaces de realizar tareas de manera eficiente y precisa. Dentro de esta área, los carritos seguidores de línea representan un caso de estudio relevante, ya que combinan control, percepción y toma de decisiones en tiempo real.

Este proyecto presenta el desarrollo de un carrito seguidor de línea, el cual emplea microcontroladores Raspberry Pi Pico y redes neuronales con aprendizaje por refuerzo para mejorar su desempeño. La capacidad de seguir una línea negra trazada en el suelo de manera autónoma se logra mediante la integración de diversos componentes y técnicas avanzadas de procesamiento de datos.

Para la percepción del entorno, se utiliza una cámara OV7670 que capta imágenes de la línea negra. Estas imágenes son procesadas en tiempo real por los microcontroladores Raspberry Pi Pico, que emplean algoritmos de visión por computadora para identificar la posición de la línea. La información procesada se utiliza para tomar decisiones de control y ajustar la trayectoria del carrito.

El aprendizaje por refuerzo juega un papel crucial en este proyecto. Este enfoque permite que el sistema aprenda y mejore su comportamiento a través de la interacción con el entorno, optimizando su capacidad para seguir la línea de manera precisa y eficiente. La red neuronal utilizada es entrenada para maximizar la recompensa, que en este caso se define como la capacidad del carrito para mantenerse sobre la línea.

El control de los motores que impulsan las ruedas del carrito se realiza mediante un módulo integrado de un puente H, que permite una regulación precisa de la velocidad y dirección de los motores. Este control es esencial para ajustar la trayectoria del carrito en respuesta a las decisiones tomadas por la red neuronal.

En resumen, este proyecto combina tecnologías avanzadas de microcontroladores, visión por camara y aprendizaje por refuerzo para desarrollar un carrito seguidor de línea autónomo y eficiente. La implementación de estos componentes no solo demuestra la viabilidad de los sistemas autónomos en tareas específicas, sino que también abre la puerta a futuras mejoras y aplicaciones en el campo de la robótica móvil.

II. Objetivos

II-A. Objetivo General

Desarrollar un carrito seguidor de línea autónomo empleando microcontroladores Raspberry Pi Pico y redes neuronales con aprendizaje por refuerzo, capaz de detectar y seguir una línea negra utilizando una cámara OV7670 y controlando los motores mediante un módulo de puente H.

II-B. Objetivos Específicos

1. Implementar la captura y procesamiento de imágenes:
   - Configurar y calibrar la cámara OV7670 para capturar imágenes de alta calidad de la línea negra en diversas condiciones de iluminación.
   - Desarrollar algoritmos de visión por computadora para procesar las imágenes en tiempo real y detectar la posición de la línea.
2. Desarrollar el modelo de aprendizaje por refuerzo:
   - Diseñar y entrenar una red neuronal que utilice aprendizaje por refuerzo para optimizar las decisiones de control del carrito.
   - Evaluar y ajustar los parámetros del modelo para mejorar la precisión y eficiencia del seguimiento de la línea.
3. Integrar y controlar los componentes del sistema:
   - Configurar los microcontroladores Raspberry Pi Pico para recibir los datos de la cámara y ejecutar el modelo de red neuronal.
   - Implementar el control de los motores utilizando un módulo de puente H para ajustar la dirección y velocidad del carrito en respuesta a las decisiones del modelo.
4. Realizar pruebas y validar el sistema:
   - Diseñar escenarios de prueba con diferentes configuraciones de líneas y curvas para evaluar el desempeño del carrito.
   - Analizar los resultados de las pruebas para identificar posibles mejoras y optimizar el comportamiento del sistema.
5. Documentar el desarrollo y resultados del proyecto:

- Elaborar una documentación detallada del diseño, implementación y pruebas del sistema.
- Presentar los resultados obtenidos y discutir las implicaciones del uso de redes neuronales y aprendizaje por refuerzo en aplicaciones de robótica móvil.

III. Desarrollo

En la realización de este proyecto se tuvieron en cuenta diferentes factores que se acomodaban a nuestro presupuesto y a nuestros conocimientos aprendidos durante cada clase de Diseño Digital Con Microcontroladores con el profesor Gerardo Muñoz de la Univeridad Distrital Fransisco José de Caldas. El objetivo era desarrollar un carrito seguidor de linea a través de una camara OV7670 sin la necesidad de usar los típicos sensores de linea, implementando los microcontroladores rasberry pi pico y rasberry pi pico w, con el el fin de usar algoritmos que permitan desarrollar redes neuronales con de aprendizajes por refuerzo en lenguajes de programación como Micropython y Circuitpython. A continuación los materiales utilizados en este proyecto.

III-A. Materiales

1. Camara OV7670.

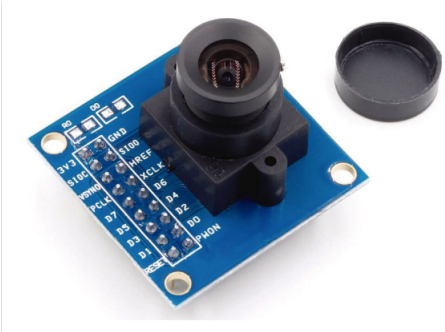


Figura 1: Camra OV7670

2. Rasbery pi pico

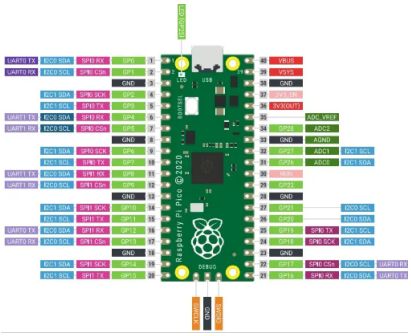


Figura 2: Rasberry pi pico

item Rasbery pi pico w

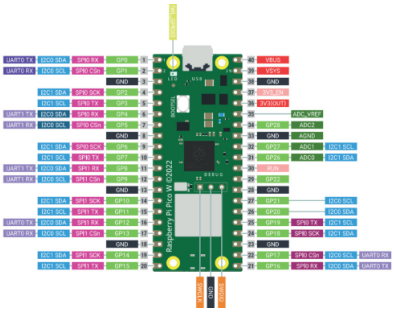


Figura 3: Rasberry pi pico w

3. Pantalla OLED 128x32

Figura 4: OLED

4. Modulo Puente H

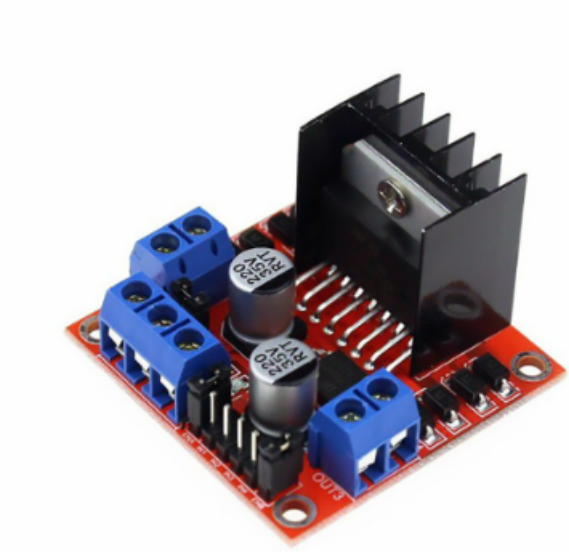


Figura 5: Diagrama de bloques

5. Jumpers de conexión

Figura 6: Jumpers

6. Carrito de tres ruedas

Figura 7: Carrito de tres ruedas

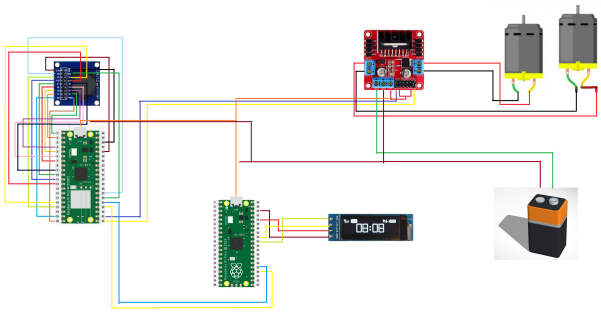
7. Bateria

Figura 8: Bateria

Además de los anteriores materiales, se utilizaron dos cables para energizar las rasberry´s con el modulo del puente H.

IV. Diseño

IV-A. Esquema de Conexiones



IV-B. Captura de imagen con 3 píxeles

Por ser la linea una imagen a blanco y negro, con rectas y 2 tipos de curvas, no es necesario utilizar los 30x40 píxeles disponibles ni todos los posibles valores de cada píxel (que representan los colores), en lugar de esto se uso 3 filas compuestas de 40 píxeles en 3 posiciones: Superior, Centro e Inferior.

Fila superior	0-18px	18-22px	22-40px
Fila centro	0-11px	11-29px	29-40px
Fila inferior	0-13px	13-25px	25-40px

Figura 9: Distribución de Filas y de píxeles

Cuando el promedio de las 3 filas de píxeles están en el centro indica que el carro esta en una linea RECTA.

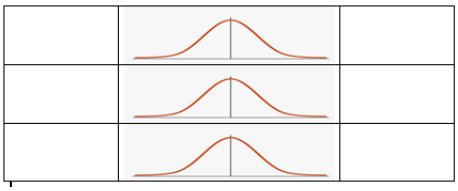


Figura 10: Promedio cuando se identifica una recta

Cuando se aproxima una curva hacia la derecha se modifican los promedios, gráficamente se ve una curva a la derecha de la siguiente forma:

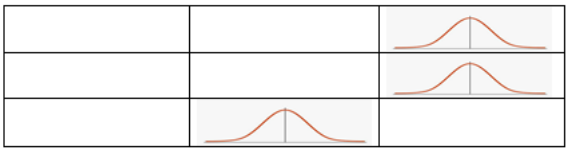


Figura 11: Promedio cuando se identifica cruva a la derecha

A continuación se presentan partes del código donde se seleccionan las filas, donde se calculan los promedios, donde se identifica la curva y como se activan los motores:

```
1
2   # Obtener la última lista de la lista de filas
3       ultima_lista = all_rows[-1]
4       central_lista = all_rows[-13]
5       primera_lista = all_rows[-24]
6
7
8   centro=UltimaImpar1[18:22] #pequeño es negro 8
    ↪   DATOS Px Arriba de la CAM
9       #print("arriba:", centro)
10      CenPro = sum(centro) / len(centro)
11      #print("Promedio del centro:", CenPro)
12      der = UltimaImpar1[13:18]
13      DerProm = sum(der) / len(der)
14      #print("Promedio del derecha:", DerProm)
15      izq = UltimaImpar1[22:27]  # Tomar los
        ↪  últimos 5 datos
16      #print("izquierda:", izq )
17      IzqProm = sum(izq) / len(izq)
```

```
    #print("Promedio del izquierda:", IzqProm )

if curvDer==1 and centro==1 and abajo==1:#solo identifica MOSTRAR EN PANTALLA
        print("--Aprox curva a la der--")
        ##mostrar_mensaje("Aprox curva der")

if CenPro3<1 and DerProm3>0:#Px ABAJO de la CAM para siempre seguir la LINEA
        pwm1.duty_cycle = 60000#20000 #activa motor derecho para centrarse
        pwm2.duty_cycle =50000 #10000
        time.sleep(0.014)
        pwm1.duty_cycle = 0
        pwm2.duty_cycle =0
        print("Activa mot derecha")
        pwmInf.duty_cycle= 0#imprimeLCD
```

IV-C. *Perceptrón desarrollado - una neurona*

Se desarrolló la siguiente neurona, ingresan 3 entradas que representan hacia donde esta corrido el promedio de cada fila de pixeles(izquierda, derecha y centro), se asignan pesos aleatorios y se genera la mejor salida posible para cada PWM.

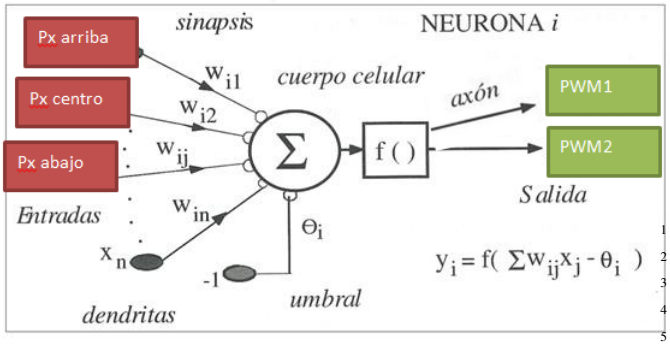


Figura 12: Diseño de perceptrón

El código desarrollado fue el siguiente, donde se observa la clase perceptron con las tres funciones basicas de inicializacion, prediccion y entrenamiento. Adicionalmente se observa la funcion de activación, la inicialización de los pesos y la tasa de aprendizaje.

```
class Perceptron:
    def __init__(self, num_inputs):
        # Inicializa los pesos y el sesgo
        self.weights = [0.0] * num_inputs
        self.bias = 0.0

    def predict(self, inputs):
        # Realiza la predicción basada en las entradas y los pesos
        activation = sum(x * w for x, w in zip(inputs, self.weights)) + self.bias
        return 1 if activation >= 0 else 0

    def train(self, training_inputs, labels, learning_rate=0.1, epochs=10):
        # Entrena el perceptrón ajustando los pesos
        for epoch in range(epochs):
            print(f"Epoch {epoch + 1}:")
            for inputs, label in zip(training_inputs, labels):
                prediction = self.predict(inputs)
                error = label - prediction
                # Actualiza los pesos basado en el error y la tasa de aprendizaje
                self.weights = [w + learning_rate * error * x for x, w in zip(inputs, self.weights)]
                self.bias += learning_rate * error
                # Imprime los pesos y el sesgo después de cada actualización
                print(f"Weights: {self.weights}, Bias: {self.bias}")
```

IV-D. *Implementación de OLED*

Para la implementación de la pantalla OLED se utilizó un pantalla de 128x32 la cual cuenta con 4 pines: VCC, GND, SDA, SCL. La cual utilizamos para indicar el valor de la velocidad de cada motor a través de un tacómetro que indica si va al máximo o si el motor se encuentra apagado. Adicionalmente se le añadió una flecha que indica hacia que dirección se dirige nuestro seguidor de linea. Para comunicar las dos rasberry´s utilizamos la interfaz I2C que nos permite enviar información de una rasberry a la otra. A continuación el código implementado en la rasberry pi pico w con circuitpython que es la encargada de dirigir nuestro seguidor de linea.

```
import board
import busio
import time

# Configuración de los pines I2C
i2c = busio.I2C(scl=board.GP19, sda=board.GP18)

# Dirección I2C del esclavo
SLAVE_ADDRESS = 0x12

# Variable a enviar
variable_a_enviar = 0  # Inicializa la variable a enviar

def enviar_dato(dato):
    # Convierte el dato a enviar a bytes
    data = bytes([dato])
    while not i2c.try_lock():
        pass
    try:
        i2c.writeto(SLAVE_ADDRESS, data)
        print(f"Enviado: {dato}")
    except Exception as e:
        print(f"Error al enviar: {e}")
    finally:
        i2c.unlock()

try:
    while True:
        # Aquí puedes cambiar el valor de variable_a_enviar según tus necesidades
```

```
        variable_a_enviar = (variable_a_enviar + 1) % 3  # Ciclo entre 0, 1, 2
        enviar_dato(variable_a_enviar)
        time.sleep(1)  # Envía el dato cada segundo
except KeyboardInterrupt:
    pass
```

Para la conexión entre la rasberry pi pico con el lenguaje de micropython y la pantalla oled 128x32, fue necesario utilizar la librería ssd1306, la cual nos permite una mejor comunicación entre la pantalla oled y nuestro microcontrolador. A continuación el código implementado:

```
import machine
import ssd1306
import time
import random
from machine import Pin, I2C

# Configuración de la pantalla OLED
i2c_oled = machine.I2C(1, scl=machine.Pin(27), sda=machine.Pin(26), freq=400000)
oled = ssd1306.SSD1306_I2C(128, 32, i2c_oled)

# Dirección I2C del esclavo
SLAVE_ADDRESS = 0x12

# Configura los pines I2C para recibir datos
i2c_direccion = I2C(0, scl=Pin(19), sda=Pin(18), freq=400000)

# Variable global para almacenar la dirección recibida
direccion_recibida = 0

# Definir las coordenadas de la barra
bar_x = 0  # Coordenada X de la esquina superior izquierda de la primera barra
bar_y = 10  # Coordenada Y de la esquina superior izquierda de la barra
bar_width = 50  # Ancho de la barra horizontal
bar_height = 10  # Altura de la barra horizontal

# Función para leer datos del maestro
def leer_datos():
    global direccion_recibida
    if SLAVE_ADDRESS in i2c_direccion.scan():  # Verifica si el esclavo está en el bus I2C
        data = i2c_direccion.readfrom(SLAVE_ADDRESS, 1)
        direccion_recibida = int.from_bytes(data, 'little')
        print(f"Recibido: {direccion_recibida}")

# Función para determinar la dirección de la flecha basada en la dirección recibida
def get_arrow_direction(direccion):
    if direccion == 0:
        return 'frente'
    elif direccion == 1:
        return 'izquierda'
    elif direccion == 2:
        return 'derecha'
    else:
        return 'frente'  # Por defecto, la flecha apuntará hacia adelante

# Función para dibujar el tacómetro en la pantalla OLED
def draw_tachometer(rpm, direccion):
    oled.fill(0)
    # Dibujar la primera barra horizontal
    bar_value = int((rpm / 8000) * bar_width)
    oled.rect(bar_x, bar_y, bar_width, bar_height, 1)
    oled.fill_rect(bar_x, bar_y, bar_value, bar_height, 1)
    # Dibujar la segunda barra horizontal
    bar_value2 = int((rpm / 8000) * bar_width)
    oled.rect(bar_x + 75, bar_y, bar_width, bar_height, 1)
    oled.fill_rect(bar_x + 75 + (bar_width - bar_value2), bar_y, bar_value2, bar_height, 1)  # Reflejo inverso
    # Mostrar el valor del RPM
    oled.text("V1", 0, 0)
    oled.text(str(rpm), 0, 25)
    # Mostrar el valor del RPM en la segunda mitad
    oled.text("V2", 110, 0)
    oled.text(str(rpm), 95, 25)
    # Determinar la dirección de la flecha
    arrow_direction = get_arrow_direction(direccion)
    # Dibujar flecha en la mitad de la pantalla
    arrow_length = 12  # Longitud de la flecha
    arrow_x = 63  # Coordenada X del centro de la flecha
    arrow_y = 28  # Coordenada Y del centro de la flecha
    if arrow_direction == 'frente':
        oled.line(arrow_x, arrow_y + arrow_length, arrow_x, arrow_y - arrow_length, 1)  # Línea vertical de la flecha
        oled.line(arrow_x, arrow_y - arrow_length, arrow_x - 4, arrow_y - arrow_length + 4, 1)  # Línea izquierda de la flecha
        oled.line(arrow_x, arrow_y - arrow_length, arrow_x + 4, arrow_y - arrow_length + 4, 1)  # Línea derecha de la flecha
    elif arrow_direction == 'derecha':
        oled.line(arrow_x - arrow_length, arrow_y, arrow_x + arrow_length, arrow_y, 1)  # Línea horizontal de la flecha
        oled.line(arrow_x + arrow_length, arrow_y, arrow_x + arrow_length - 4, arrow_y - 4, 1)  # Línea superior de la flecha
        oled.line(arrow_x + arrow_length, arrow_y, arrow_x + arrow_length - 4, arrow_y + 4, 1)  # Línea inferior de la flecha
    elif arrow_direction == 'izquierda':
```

```
        oled.line(arrow_x + arrow_length, arrow_y, arrow_x - arrow_length, arrow_y, 1)  # Línea horizontal de la flecha
        oled.line(arrow_x - arrow_length, arrow_y, arrow_x - arrow_length + 4, arrow_y - 4, 1)  # Línea superior de la flecha
        oled.line(arrow_x - arrow_length, arrow_y, arrow_x - arrow_length + 4, arrow_y + 4, 1)  # Línea inferior de la flecha
    oled.show()

try:
    while True:
        leer_datos()  # Leer datos del maestro
        rpm = get_random_rpm()
        draw_tachometer(rpm, direccion_recibida)
        # Actualizar cada segundo, puedes ajustar esto según tu preferencia
        time.sleep(1)
except KeyboardInterrupt:
    pass
```

IV-E. Aprendizaje por refuerzo

Se utilizo la estrategia de Q-Learning que utiliza los píxeles de la cámara como entradas para el estado. Se ajusta los valores Q en base a la recompensa recibida y las expectativas futuras, permitiendo al agente aprender a maximizar sus recompensas.[a] lo largo del tiempo. Es un enfoque de premio castigo, es decir cuando sigue la linea obtiene mayor recompensa. Para este caso el entorno es la "pista"que tiene rectas y curvas, el agente esta representada por la clase QLearnigAgent donde se observan las funciones de seleccionar acción, obtener un estado del entorno y las recompensas (reward en la función update).

Es sencillo entender este modelo con la siguiente imagen:

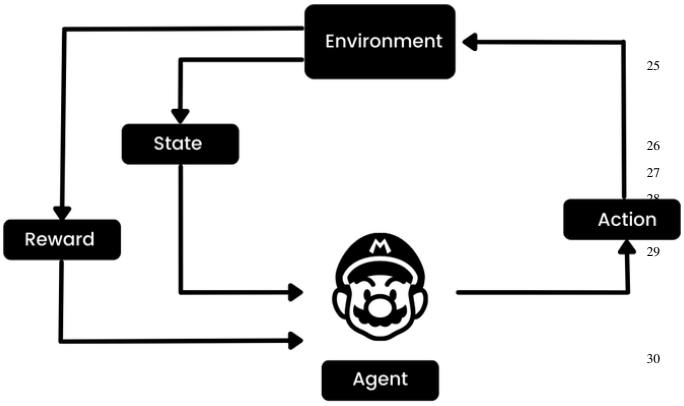


Figura 13: Modelo Q-Learning

El código para realizar la implementación de este modelo, fue el siguiente utilizando una nueva clase QLearningAgent:

```
######## Aprendizaje por refuerzo
class QLearningAgent:
    def __init__(self, num_states, num_actions, alpha=0.1, gamma=0.9, epsilon=0.1):
        # Inicialización del agente Q-learning con los parámetros dados
        self.num_states = num_states    # Número de posibles estados
        self.num_actions = num_actions # Número de posibles acciones
        self.alpha = alpha              # Tasa de aprendizaje
        self.gamma = gamma              # Factor de descuento
        self.epsilon = epsilon
        self.q_table = np.zeros((num_states, num_actions))

    def choose_action(self, state):
        # Selección de una acción basada en la política epsilon-greedy
        if random.uniform(0, 1) < self.epsilon:
            return random.randint(0, self.num_actions - 1)
        else:
            # Explotación: selecciona la acción con el mayor valor Q para el estado dado
            return np.argmax(self.q_table[state])

    def update(self, state, action, reward, next_state):
        # Actualización de la tabla Q usando la fórmula de Q-learning
        best_next_action = np.argmax(self.q_table[next_state])  # Mejor acción en el próximo estado
        td_target = reward + self.gamma * self.q_table[next_state, best_next_action] # Objetivo de TD (Temporal Difference)
        td_error = td_target - self.q_table[state, action] # Error de TD
        self.q_table[state, action] += self.alpha * td_error # Actualización de la tabla Q

    def get_state(self, pixels):
        # Lógica para convertir los píxeles en un estado
        binary_pixels = [1 if pixel < 100 else 0 for pixel in pixels] # Binariza los píxeles (1 si el valor es menor a 100, 0 en caso contrario)
        state = sum([binary_pixels[i] << i for i in range(len(binary_pixels))]) # Convierte la lista de píxeles binarios en un único estado entero
        return state
```

V. Problemas identificados

1. Intensidad de la luz La intensidad de la luz que recibía la cámara variaba en función de diferentes factores, como la distancia entre el carro y la línea, la obstrucción de la luz por objetos o sombras, o cambios en la iluminación del ambiente.

Estas variaciones pueden afectaron la precisión de la cámara y causó que el seguidor de línea desviara de su trayectoria.

2. Velocidad de los motores Si los motores del seguidor de línea no estaban sincronizados correctamente, es decir, si uno gira a una velocidad mayor que el otro, el carro tendía a desviarse de la línea y en ocasiones perdía la pista por completo. Esto puede deberse a variaciones en la calidad de los motores o a un mal ajuste de los parámetros.
3. Ángulo de la cámara Cuando el ángulo de la cámara no era el adecuado, los datos de la línea, no se capturaban de forma correcta. Esto provocó que el seguidor de línea no fuese capaz de mantenerse en el camino correcto y se desvíe de su trayectoria. .
4. Altura de la cámara Cuando la cámara no se encontraba a una altura adecuada con respecto al suelo, los datos no se capturaban con precisión en la línea, especialmente si esta presentaba curvas pronunciadas o cambios bruscos de dirección. Por lo que el seguidor de línea no seguía la ruta marcada.
5. Diferentes tipos de líneas Uno de los principales problemas que se nos presentó fue la variabilidad en los tipos de líneas a seguir. Pues el valor de los datos que tomamos, cambiaban según fuese el tamaño de la línea. .
6. Descarga rápida de la batería Al principio se utilizaron baterías de 9V, pero se descargaban muy rápido. Esto puede deberse a un consumo excesivo de energía por parte del carro, causando que la batería se agote rápidamente y afectando su autonomía.
7. Peso sobre el carro A medida que fuimos implementando componentes en el carro, el peso que este debía soportar aumentaba, lo que causaba que las ruedas no funcionaran correctamente e incluso su velocidad disminuyera.

VI. Mejoras realizadas

Teniendo en cuenta los problemas identificados a lo largo del diseño e implementación del carro, se toman los mismos ítems para poder explicar las mejoras que se realizaron con el fin de solucionarlos.

1. Intensidad de la luz Para solucionar este problema, se hizo mejoras en los valores de los rangos que se estaban tomando al leer la línea, con varias pruebas y sesiones de entrenamiento.
2. Velocidad de los motores Se realizaron varias pruebas con el fin de llegar a una velocidad que fuera lo suficientemente rápida en las rectas, pero que a la vez, le diera tiempo de detectar las curvas y saber a cuál motor aumentarle y disminuirle la velocidad.
3. Ángulo de la cámara Se realizaron varias pruebas para poder definir que ángulo era el más apropiado para poder tener una buena visión de la línea.
4. Altura de la cámara Se realizaron varias pruebas con el fin de definir la altura adecuada y tener una buena visión de la línea y que diera el tiempo de detectar las curvas.
5. Diferentes tipos de líneas Después de realizar las pruebas pertinentes con las líneas, se llegó al promedio del valor de los datos para la línea de prueba y no tener inconvenientes.
6. Descarga rápida de la batería Luego de tener en funcionamiento el carro con diversas baterías, se optó por utilizar una batería de dron, para asegurarnos de tener una buena alimentación para el carro.
7. Peso sobre el carro Al ver que había mucho peso sobre el carro, se realizó una mejor distribución del peso de los componentes, pues el problema que se presentaba, se debía a que la mayor acumulación de peso estaba en un solo lugar.

VII. Resultados obtenidos

Después hacer todas las mejoras que nuestro diseño requería, se puede decir que se logró un muy buen tiempo de carrera, pues tarda 56 segundos en recorrer la pista propuesta, también se observa una constante corrección del recorriendo con los motores, es decir, la cámara detecta rápidamente cuando está desviado del camino y se aumenta la velocidad del motor requerido. Aunque nuestro carro no es tan rápido, se puede observar que avanza de forma constante lo que nos permite contar con un buen tiempo de recorrido. Para finalizar, se observa que en las últimas, se estaba detectando la línea de forma correcta y ya no ocurrían desvíos repentinos.

VIII. Link código completo en GitHub y Link de vídeo explicativo

VIII-A. Código completo del proyecto en Github

https://github.com/DarioSanabriaUD/ActividadesMicros/blob/fc967731bc95a1d9

VIII-B. Link de vídeo explicativo

IX. Conclusiones

1. Cuando se trabaja con un sistema de aprendizaje por refuerzo, como en este caso, es importante realizar la mayor cantidad de pruebas posibles, pues de esta manera se logra que el robot tenga la mayor cantidad escenarios posibles y aprender a solucionarlos.
2. De la misma forma nosotros como los diseñadores, haciendo tantas pruebas como sea posible, podemos detectar errores y corregirlos lo antes posible en la parte del hardware y el software.
3. Es importante resaltar que tanto al hardware como el software es muy importante, es decir, podemos contar con el mejor de los software, pero si nuestro hardware no es el adecuado, difícilmente el proyecto va a salir como se espera, esto podemos notarlo en el momento en l que se trabajó con baterías 9V, pues la alimentación no era la que se requería, por lo que tuvimos que recurrir a una batería más potente.
4. Después de bastantes pruebas y varias correcciones de problemas, logramos tener un carro seguidor de linea completamente funcional, además de contar con un muy buen tiempo de recorrido de pista.