Informe final: Carro seguidor de linea

Jaider Alejandro Soto Ovalle - 20202005015
Joshua David Cantillo Latorre - 20201005101
Jhon Alexsander Moreno Velasquez - 20202005031
jasotoo@udistrital.edu.co
jdcantillol@udistrital.edu.co
jhamorenov@udistrital.edu.co
Universidad Distrital Francisco José De Caldas
Facultad de Ingeniería
Proyecto Curricular Ingeniería Electrónica
Bogotá D.C. - Colombia

I. Introducción

En este informe se presenta el desarrollo de un proyecto de diseño de microcontroladores enfocado en la creación de un carro seguidor de línea. El proyecto se llevó a cabo en tres fases principales.

Inicialmente, se diseñó un sistema de control remoto que permitía manejar el carro a través de una conexión Wi-Fi. Este sistema estableció una comunicación robusta entre el usuario y el vehículo, permitiendo un control manual eficiente.

En la segunda fase, se integró una cámara OV7670 al carro para capturar imágenes del entorno. Estas imágenes se procesaron para detectar y seguir una línea en el suelo, utilizando un algoritmo basado en el cálculo de errores. Esto permitió que el carro ajustara su trayectoria automáticamente para mantenerse sobre la línea.

En la última fase, se implementó una red neuronal, específicamente un perceptrón, que aprendió del comportamiento del sistema de seguimiento de línea. Esta red neuronal fue entrenada para mejorar la precisión y eficiencia del seguimiento, adaptándose y optimizando continuamente el control del carro.

Este informe abarca los detalles técnicos de cada fase del desarrollo, los retos enfrentados y las soluciones aplicadas. Se analizan los resultados obtenidos y se discuten las mejoras alcanzadas con la implementación del perceptrón, demostrando la evolución del sistema desde un control manual hasta un seguidor de línea autónomo y optimizado mediante aprendizaje automático.

II. Objetivos

- Describir la implementación del control remoto por Wi-Fi y su funcionamiento en el manejo del carro.
- Explicar el desarrollo del seguidor de línea utilizando la cámara OV7670 y el algoritmo basado en el cálculo de errores.
- Desarrollar y evaluar una red neuronal (perceptrón) para mejorar la precisión y eficiencia del seguidor de línea.
- Documentar los resultados obtenidos en cada fase del proyecto y discutir las mejoras logradas con la implementación del perceptrón.

III. Materiales

Los materiales que se utilizaron para este proyecto se relacionan a continuación:

Figure 1: Chasis de carro

Un chasis para carros es una estructura ligera y duradera, diseñada para montar motores, ruedas, sensores y placas de control como Raspberry Pi. Facilita la instalación de componentes y asegura estabilidad y maniobrabilidad para proyectos robóticos.

Figure 2: Motoreductores

Dispositivos que combinan un motor eléctrico con una caja de engranajes para reducir la velocidad y aumentar el torque de salida. Son ampliamente utilizados en proyectos de electrónica, robótica y mecatrónica para proporcionar potencia y control de movimiento a diferentes aplicaciones.

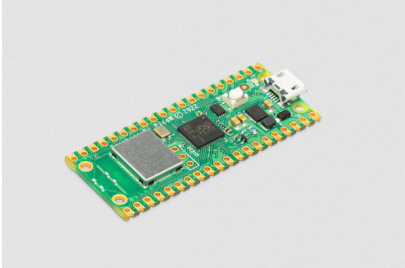

Figure 3: Raspberry pi pico W

El Raspberry Pi Pico es un microcontrolador de bajo costo desarrollado por la Fundación Raspberry Pi. Integra un potente procesador ARM Cortex-M0+ de doble núcleo, lo que lo hace ideal para una amplia gama de proyectos de electrónica. Además, cuenta con una variedad de puertos de entrada/salida (GPIO) que permiten la conexión de sensores, actuadores y otros dispositivos externos.

Figure 4: Camara OV7670

Es ampliamente utilizada en proyectos de electrónica y robótica debido a su bajo costo y su capacidad para capturar imágenes y videos en tiempo real. La OV7670 es capaz de capturar imágenes con una resolución de hasta 640x480 píxeles (VGA) y puede transmitir datos de imagen a través de interfaces como SCCB (Serial Camera Control Bus) y paralela.

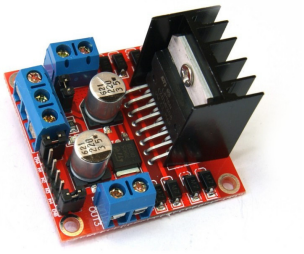

Figure 5: Puente h L298

Proporciona una forma eficiente de invertir la dirección de rotación y controlar la velocidad de los motores. El L298 incluye dos puentes H, cada uno capaz de controlar un motor, y puede manejar corrientes de hasta 2 amperios por puente.

Figure 6: Pantalla OLED

Una pantalla OLED con comunicación I2C utiliza el protocolo I2C para interactuar con otros dispositivos, como microcontroladores. Esto simplifica la conexión y permite mostrar información en la pantalla de manera eficiente.

Figure 7: Fuente de alimentacion

IV. Diseño del Proyecto

A través de diversas fases de desarrollo, desde la implementación de un control remoto por Wi-Fi hasta la integración de una red neuronal para optimizar el seguimiento, el objetivo principal ha sido diseñar un sistema autónomo capaz de seguir una línea en el suelo. Este informe presenta el proceso de diseño, los desafíos encontrados y las soluciones implementadas para lograr un seguimiento preciso y eficiente.

A. Primera Propuesta: Carro usando control con web server:

La primera propuesta de nuestro proyecto de diseño de microcontroladores consistió en desarrollar un carro controlado mediante un servidor web, utilizando una conexión Wi-Fi. Esta fase inicial del proyecto sentó las bases para la posterior implementación de un sistema autónomo de seguimiento de línea. El objetivo principal fue establecer una comunicación remota confiable y eficiente que permitiera controlar el movimiento del carro a distancia a través de una interfaz web accesible desde cualquier dispositivo con conexión a Internet.

En esta sección del informe, se detallará el proceso de diseño e implementación de esta primera fase, abordando aspectos clave como la configuración del servidor web en el microcontrolador, el desarrollo de la interfaz de usuario y la integración de los sistemas de control de motores y comunicación Wi-Fi.

Para proceder con el analisis del codigo implementado, discutiremos por partes el funcionamiento del mismo.

```
input1_1 = Pin(16, Pin.OUT)
input1_2 = Pin(17, Pin.OUT)

input2_1 = Pin(18, Pin.OUT)
input2_2 = Pin(19, Pin.OUT)

# Define los pines PWM para controlar la velocidad de los motores
pwm_pin1 = Pin(20)
pwm1 = PWM(pwm_pin1)
pwm1.freq(1000)  # Frecuencia del PWM (Hz)

pwm_pin2 = Pin(21)
pwm2 = PWM(pwm_pin2)
pwm2.freq(1000)  # Frecuencia del PWM (Hz)

adc = ADC(0) #pin GP26
timer = Timer()
```

Listing 1: Declaracion de pines

En esta parte podemos ver que pines se definen como PWM, los cuales se encargaran de manejar la velocidad de los motores. También se define un pin como ADC, el cual nos ayudara a recibir el margen de error y procesarlo de manera digital, este proceso necesita adicionalmente un condensador y resistencia que va directamente conectada a la raspberry que transmite el error por medio de PWM.

```
def control_motor(input1, input2, pwm, speed, direction):
    pwm.duty_u16(int(65535 * speed/100))  # Establece el ciclo de trabajo del PWM
    if direction == 1:
        input1.on()
        input2.off()
    elif direction == -1:
        input1.off()
        input2.on()
    else:
        input1.off()
        input2.off()
```

Listing 2: Ciclo util y variable direction

En esta parte del programa, configuramos el ciclo util correspondiente para los PWM que controlaran los motores junto con una variable llamada direction, que nos facilitara invertir el sentido de giro de los motores según sea conveniente.

```
def forward(speed1, speed2):
    control_motor(input1_1, input1_2, pwm1, speed1, 1)
    control_motor(input2_1, input2_2, pwm2, speed2, 1)

def backward(speed1, speed2):
    control_motor(input1_1, input1_2, pwm1, speed1, -1)
    control_motor(input2_1, input2_2, pwm2, speed2, -1)

def lef(speed1, speed2):
    control_motor(input1_1, input1_2, pwm1, speed1, 1)
    control_motor(input2_1, input2_2, pwm2, speed2, 1)

def rigt(speed1, speed2):
    control_motor(input1_1, input1_2, pwm1, speed1, 1)
    control_motor(input2_1, input2_2, pwm2, speed2, 1)


def stop():
    control_motor(input1_1, input1_2, pwm1, 0, 0)
    control_motor(input2_1, input2_2, pwm2, 0, 0)
```

Listing 3: Funciones para el manejo de los motores

Esta parte del código describe las diferentes funciones creadas para controlar el carro, realizando movimientos hacia diferentes lugares. Las funciones se llaman con ayuda de la pagina web html que se explicara mas detalladamente mas adelante en el informe.

```
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
# See the MAC address in the wireless chip OTP
mac = ubinascii.hexlify(network.WLAN().config('mac'),':').decode()
print('mac = ' + mac)
ssid = "redwifi"
pw = "stks6105"
wlan.connect(ssid, pw)
# Wait for connection with 10 second timeout
timeout = 10
while timeout > 0:
    if wlan.status() < 0 or wlan.status() >= 3:
        break
    timeout -= 1
    print('Waiting for connection...')
    time.sleep(1)
```

Listing 4: Configuracion wifi

Este código configura la conexión Wi-Fi en un dispositivo, imprime su dirección MAC, y trata de conectarse a una red específica usando un SSID y contraseña dados. Luego, espera hasta que la conexión se establezca o hasta que pasen 10 segundos. Durante este tiempo, imprime mensajes indicando que está esperando la conexión. Una vez que la conexión se establece o el tiempo límite se alcanza, el programa termina.

```
def get_html(html_name):
    with open(html_name, 'r') as file:
        html = file.read()

    return html
# HTTP server with socket
addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]
s = socket.socket()
s.setsockopt(socket.SOL_SOCKET, socket.
    SO_REUSEADDR, 1)
s.bind(addr)
s.listen(1)
print('Listening on', addr)
led = machine.Pin('LED', machine.Pin.OUT)
```

Listing 5: Archivo HTML y servidor

Este código define una función para obtener el contenido de un archivo HTML y luego establece un servidor HTTP en el puerto 80 para manejar solicitudes entrantes, mientras inicializa un pin de salida para un LED. Posteriormente se llaman todas las funciones mencionadas y se ejecutan de manera que en un ciclo while true se pueda controlar el carro. A se explica un poco el archivo html que contiene el diseño de la pagina web y por ende del control remoto.

```
</head>
<body>
    <div id="arm">
        <h1>Jaider Soto, Jhon Moreno, Joshua</
            h1>
        <p>Control Remoto</p>
        <p>Valor ADC: {adc_value}</p> <!--
            Marcador de posici n -->
        <div class="button-container">
            <button onclick="sendRequest('?
                motor=forward')" class="button"
                >   </button>
        </div>
        <div class="button-container">
            <button onclick="sendRequest('?left
                =hard')" class="button">   </
                button>
            <button onclick="sendRequest('?
                motor=stop')" class="button
                stop">STOP</button>
            <button onclick="sendRequest('?
                rigth=hard')" class="button">
                   </button>
        </div>
        <div class="button-container">
            <button onclick="sendRequest('?left
                =left')" class="button">   </
                button>
            <button onclick="sendRequest('?
                rigth=rigth')" class="button">
                   </button>
        </div>
        <div class="button-container">
            <button onclick="sendRequest('?
                motor=backward')" class="button
                ">Back</button>
        </div>
        <div class="button-container">
            <button onclick="sendRequest('?
                quite=quite')" class="button">
                Quit</button>
        </div>
    </div>
</body>
</html>
```

Listing 6: Diseño pagina web

Este es un archivo HTML que crea una interfaz web para un control remoto. Incluye botones para controlar un motor en diferentes direcciones, como hacia adelante, atrás y hacia los lados. También hay un botón de parada y un marcador de posición para mostrar el valor del convertidor analógico-digital (ADC). Además, utiliza JavaScript para enviar solicitudes HTTP al servidor cada vez que se presiona un botón, lo que permite controlar el dispositivo conectado a través de la interfaz web. A continuación se enseña el diseño estético final de esta:

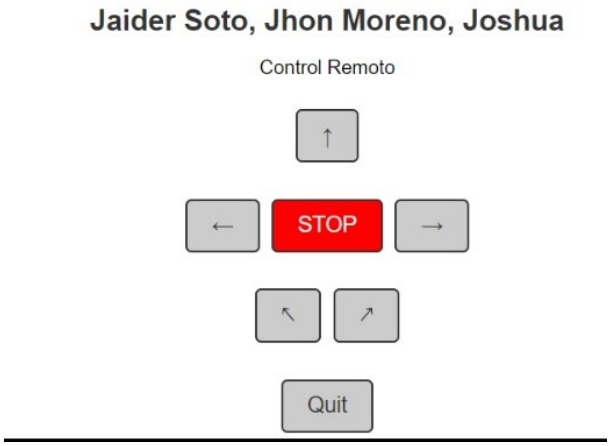

Figure 8: Diseño control remoto

B. Segunda Propuesta: Seguidor de linea autónomo con la OV7670

En esta nueva fase, se hace énfasis en mejorar el seguimiento de la línea por parte del carro , por lo que se hace un análisis de los pro y contra que tiene el uso de un control por protocolo WIFI para realizar el seguimiento de la línea, de los cuales cabe destacar los principales inconvenientes, como por ejemplo:

- Se produce un retardo en la asignación de una acción mediante el control del dispositivo móvil y la respuesta a esta por parte del carro.
- El posible error humano que puede generarse al tratar de seguir la línea y su ajuste mediante los

controles.

- El ajuste es demasiado brusco, lo que produce una respuesta fuera de lo esperado.

Por estos motivos, entre otros, se llegó a la conclusión de que la manera más efectiva de recopilar los datos que serán utilizados en una fase posterior para el entrenamiento de la red neuronal, es lograr un control autónomo del carro en base a los datos recopilados por la cámara.

Para dicho proposito se tomó como referencia el siguiente repositorio de GitHub https://github.com/GerardoMunoz/Curso_Python/blob/main/ejem/ov7670_ejem.py en el que se hace una aplicación en circuitpython para visualizar en cáracteres ASCII los datos capturados por una cámara OV7670 e imprimirlos en la consola.

Los cambios que se aplican a dicho código tomado del repositorio se mencionan a continuación.

```
cam.capture(buf)

matrix = []
rw = []

for i in range(cam.width):
    intensity = 0 if buf[2 * (width * (cam.
        height-1) + i)] * 10 // 255 < 5 else 1
    rw.append(intensity)

matrix.append(rw)

    ulinea=matrix[len(matrix)-1][:]
```

Listing 7: Cambios en la captura de la imagen

En el fragmento de código presentado, se toma el buffer que contiene los datos capturados por la cámara en un espacio de color YUV , para binarizarlos ,considerando como un 0 aquellos datos en la intensidad que sean menores a 5 y un 1 aquellos que sean mayores ,esto nos permite hacer una distinción muy aproximada de la línea negra sobre el fondo blanco.

Sin embargo, no se trabaja sobre todo el conjunto de datos conteniedos por el buffer, por motivos de eficiencia se opta por utilizar unicamente la última fila capturada por este, la cuál es almacenada en un arreglo llamado ulinea.

C. Procesamiento de datos

Con los datos almacenados en ulinea, se procede a hallar la manera más efectiva para mantener el centro de la línea ,por lo que primero , se debe hallar dicho centro en relación al centro de la cámara, para dicho fin se realiza el cálculo de la posición promedio de los ceros en ulinea y se compara con la mitad de la longitud de dicha lista.Con este valor de la posición promedio y el centro de la cámara, se puede hallar el valor de error que se presenta entre estos, el cuál representará un desfase hacia la izquierda o derecha del carro respecto a la línea, siendo dicho error (diferencia) positivo en caso de un desfase hacia la derecha y negativo si se presenta un desfase hacia la izquierda como se presenta a continuación.

```
suma=0
contador=0
for i in range(len(ulinea)):
    suma += i if ulinea[i]==0 else 0
    contador += 1 if ulinea[i]==0 else 0
    promedio=suma/contador if contador!=0 else
        0
diferencia=(20-promedio)*(100/20) if promedio
    !=0 else 100
```

Listing 8: Cálculo del centro de la línea

Dicha diferencia se puede utilizar para varíar los valores de los PWM de los motores en función de ésta, para lo cual, al presentarde una diferencia negativa se procederá a disminuir la velocidad del motor derecho en el mismo porcentaje que la magnitud de diferencia, mientras que si se presenta una diferencia positiva ,será el motor izquierdo quien deberá modificar el valor de su PWM en base a ésta.

```
if promedio != 0 :
    if diferencia<0:
        print(abs(diferencia/100))
        pwm_derecha.duty_cycle = int(((
            vel_ini_derecha/100)*65535) * (1-
            abs((diferencia)/100)))
        pwm_izquierda.duty_cycle = int((
            vel_ini_izquierda/100)*65535)
    else:
        pwm_izquierda.duty_cycle = int(((
            vel_ini_izquierda/100)*65535) * (1-
            abs((diferencia)/100)))
        pwm_derecha.duty_cycle = int((
            vel_ini_derecha/100)*65535)
else:
    pwm_derecha.duty_cycle = 0
    pwm_izquierda.duty_cycle = 0
```

Listing 9: Ajuste de los PWM en base a la diferencia

Con esto el carro está en la capacidad de seguir la línea por su propia cuenta sin intervención humana, sin embargo nos encontramos con un error al realizar las pruebas ,el carro no realizaba ninguna acción. Luego de analizar paso a paso el código ,nos dimos cuenta de que el valor de los PWM cambia en función a cada captura realizada por la cámara ,lo que no le permitía a los motores establecerse en una velocidad ,por tal motivo se opta por agregar un tiempo de espera antes de realizar la siguiente captura con el fin de permitir a los motores tomar la velocidad que se les ha asignado.

```
time.sleep(0.06)
```

```
2
3 pwm_derecha.duty_cycle = 1000
4 pwm_izquierda.duty_cycle = 1000
```

Listing 10: Solución alta tasa de refresco del PWM

Además de la espera se decide restablecer el valor de los PWM a 1000 con el fin de que entre fotogramas el carro no mantuviera su velocidad y saliera de la pista.
Con los cambios presentados anteriormente se logró que el carro siguiera la pista por su propia cuenta.

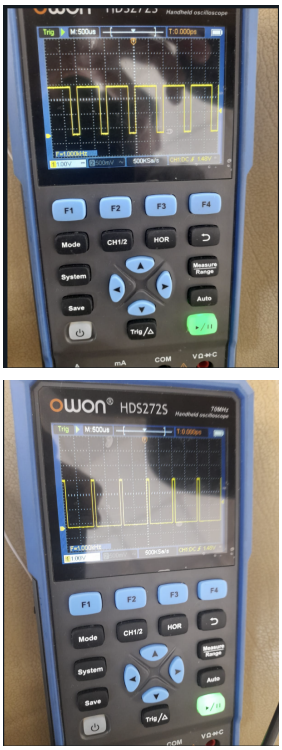

Figure 9: Cambios en los PWM según la camara

D. *Inclusión del perceptron*

En la fase anterior se logró un control autónomo del carro en base al error calculado, en esta fase, se implementarán las clases Perceptron y Matrix tomadas del repositorio guía del curso (https://github.com/GerardoMunoz/simulador), para lo cuál se debe realizar la importación de dichas clases al código de nuestro carro y además realizar una instancia de la clase Perceptron antes del bucle principal del código.

```
1 from Matrix import Matrix
2 from Perceptron import Perceptron
3
4 perceptron = Perceptron(40,2)
5 #40 es el n mero de datos en ulinea y 2 la
      cantidad de PWM's
```

Listing 11: Clases Perceptron y Matrix

Una vez instanciada la clase Perceptron podemos utilizar este objeto para acceder sus métodos train y predict,por lo cuál primero debemos ajustar los datos que le enviaremos a dicho objeto.

```
1 inputs = Matrix(1, len(ulinea), ulinea)
2 labels = Matrix(1, 2, [pwm_derecha.duty_cycle ,
      pwm_izquierda.duty_cycle])
3 perceptron.train(inputs, labels,epochs=1)
```

Listing 12: Ajuste inputs y labels

En el fragmento de código anterior, se toman los datos de ulinea y los PWM de los motores para asignarlos como los datos de entrada y salida respectivamente para el entrenamiento del perceptron, por lo cual se utiliza la clase Matrix para ajustarlos a un formato correcto para el perceptron.

Una vez asignados los inputs y labels para el entrenamiento ,se procede a iniciar con éste, entrenando 1 época por cada fotograma capturado.

E. *Cambio de conducción por controlador a conducción autónoma con perceptron*

Para poder utilizar el control aprendido por el perceptron se implementa un temporizador que se encargará de cambiar entre el estado de entrenando a predecir.

```
1 wait_time = 20
2 start_time = time.monotonic()
3 entrenar=True
```

Listing 13: Creación del temporizador

Se debe agregar el código anterior antes del bucle principal del carro para poder inicializar el temporizador.Luego, se debe agregar el siguiente fragmento dentro del bucle principal para evaluar el momento en que el temporizador supere el tiempo asignado ,que en este caso es $wait_time = 20$.

```
1 current_time = time.monotonic()
2 elapsed_time = current_time - start_time
3
4 if elapsed_time >= wait_time:
5     entrenar=False
```

Listing 14: Evaluación del temporizador

Como se aprecia, una vez transcurrido el tiempo asignado, se ejecuta el código dentro del condicional, el cual asigna a la variable entrenar el valor de *False*. Ahora debemos crear un condicional que permita evaluar el valor de *entrenar*. dicho condicional se presenta a continuación.

```
1 if(entrenar==True):
2     #Se ejecuta el c digo ya visto de
          conducci n en base al error calculado
          y se entrena el perceptron
```

```
else:
    prediction = perceptron.predict(inputs)
    if (int(prediction[0, 0]) and int(
        prediction[0, 1])) <= 65535:
        pwm_derecha.duty_cycle=abs(int(
            prediction[0, 0]))
        pwm_izquierda.duty_cycle=abs(int(
            prediction[0, 1]))
    time.sleep(0.05)
    pwm_derecha.duty_cycle = 1000
    pwm_izquierda.duty_cycle = 1000
```

Listing 15: Condicional de cambio de conducción

Este código permite cambiar entre entrenar el perceptron a utilizar el control de los PWM aprendido por el perceptron, para lo cual se utiliza el método predict del objeto perceptron el cual retorna una matriz con los dos valores de los PWM de los motores. Al igual que con el entrenamiento del perceptron , se agrega una espera y se restablecen los valores de PWM para permitir que se reflejen los cambios en las velocidades de los motores.

Con esto el entrenamiento de la red neuronal que generará los PWM de los motores en base a los datos obtenidos por la cámara en ulinea estaría completo. Sin embargo en otra sección se mencionarán algunos de los cambios realizados para mejorar aún más la eficiencia del entrenamiento.

V. Mejoras del diseño y el rendimiento

A. *Mejoras en el código*

A continuación se mencionan las mejoras y cambios realizados al código para lograr una mayor eficiencia.

- Se redujo el valor de diferencia a un 80% de su valor con el fin de lograr una transición más fluida entre los cambios de dirección.

```
if promedio != 0 :
    if diferencia<0:
        print(abs(diferencia/100))
        pwm_derecha.duty_cycle = int(((
            vel_ini_derecha/100)*65535) * (1-
            abs((diferencia)/100)*0.8))
        pwm_izquierda.duty_cycle = int((
            vel_ini_izquierda/100)*65535)
    else:
        pwm_izquierda.duty_cycle = int(((
            vel_ini_izquierda/100)*65535) *
            (1-abs((diferencia)/100)*0.8))
        pwm_derecha.duty_cycle = int((
            vel_ini_derecha/100)*65535)
else:
    pwm_derecha.duty_cycle = 0
    pwm_izquierda.duty_cycle = 0
```

Listing 16: Reducción valor de diferencia

B. *Uso de la OLED y conexión UART:*

La implementación de una pantalla OLED 0,96” con controlador SSD1306 se realizó usando comunicación I^2C, entonces se planteó en primera instancia mandar las velocidades desde la raspberry pi pico que controlaba la cámara, a otra raspberry la cual se encargaría de tomar esos datos y mostrarlos en la pantalla, para realizar esta transferencia de datos se optó usar la conexión UART debido a sus simplicidad y flexibilidad a la hora de implementarse, a continuación se va mostrar el código del microcontrolador que realiza la transmisión de las velocidades:

```
import sys
import time
import pwmio
import digitalio
import busio
import board

vel_ini_derecha=75;
vel_ini_izquierda=75;
uart = busio.UART(board.GP16, board.GP17,
    baudrate=9600)
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT
pwm_derecha = pwmio.PWMOut(board.GP20,frequency
    =100000, duty_cycle=0)
pwm_izquierda = pwmio.PWMOut(board.GP21,
    frequency=100000, duty_cycle=0)
```

Listing 17: Inicialización de los pines del UART

Aquí en este fragmento del código se puede ver en la línea 10 se utilizan los pines GP16 y GP17 para la comunicación UART. Después se asignan los pines que controlan los motores izquierdo y derecho, seguido de eso se crea la función que va a enviar los datos deseados a la otra raspberry:

```
# Configurar el pin como salida
motor_i = digitalio.DigitalInOut(
    pin_motor_izquierda)
motor_i.direction = digitalio.Direction.OUTPUT

# Establecer el pin en alto
motor_i.value = True

luz = digitalio.DigitalInOut(pin_ledblanco)
luz.direction = digitalio.Direction.OUTPUT
luz = True

# Establecer el pin en alto
motor_d.value = True

def enviar_texto(texto):
    uart.write(texto)
```

Listing 18: Función para enviar los datos

En este caso la función va recibir un parámetro que es el texto que queremos transmitir a la otra raspberry.

```
# Crear el texto con las velocidades de los
    motores
    texto_a_enviar = "Vel motor 1 = {}\nVel
        motor 2 = {}".format(pwm_derecha.
        duty_cycle, pwm_izquierda.duty_cycle)
    # Enviar el texto
```

```
    enviar_texto(texto_a_enviar)
    texto_a_enviar = ""
    # Enviar el texto
    enviar_texto(texto_a_enviar)
    time.sleep(0.01)  # Puedes ajustar el
        intervalo de env o seg n tus
        necesidades
```

Listing 19: Uso del funcion de enviar texto

Esta es ya la implementación de la función que envia los datos deseados a otra raspberry, se utiliza la variable *textoaenviar* que sera el parámetro que le entre a la función, en este caso son los pwm que están ajustando la velocidad de los motores. El código de la raspberry que se encarga de recibir dichos datos y mostrarlos en la pantalla es:

```
import board
import busio
import digitalio
import time
import adafruit_ssd1306

# Configurar la comunicaci n UART (TX: Pin 4,
    RX: Pin 5)
uart = busio.UART(board.GP4, board.GP5,
    baudrate=9600)

# Configurar el pin LED
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

# Configurar la comunicaci n I2C para la
    pantalla OLED
i2c = busio.I2C(board.GP17, board.GP16)
oled = adafruit_ssd1306.SSD1306_I2C(128, 32,
    i2c)

# Loop principal
while True:
    try:
        # Leer el mensaje completo desde UART
        mensaje1 = uart.readline()
        mensaje2 = uart.readline()

        # Verificar si se recibi  un mensaje
        if mensaje1 is not None and mensaje2 is
             not None:
            # Imprimir los mensajes en la
                consola
            print("Mensaje 1:", mensaje1)
            print("Mensaje 2:", mensaje2)

            # Limpiar la pantalla OLED
            oled.fill(0)

            # Dibujar los textos en la pantalla
                 OLED
            oled.text(mensaje1.decode('utf-8').
                strip(), 0, 0, 1)
            oled.text(mensaje2.decode('utf-8').
                strip(), 0, 10, 1)  # Colocar
                el segundo dato debajo del
                primero

            # Actualizar la pantalla OLED con
                los textos
            oled.show()

            # Encender el LED al recibir datos
            led.value = True

            # Esperar un momento antes de
                apagar el LED
            time.sleep(0.1)
            led.value = False
    except KeyboardInterrupt:
        break

# Cerrar la conexi n UART al salir
uart.deinit()
```

Listing 20: Código de la recepcion de los datos del UART

Aquí se puede ver que para la recepción de los datos del UART se utilizaron los pines GP4 (Tx) y GP5 (Rx), mientras que para la comunicación I^2 con la OLED se usaron los pines GP16 (SDA) y GP17 (SCL), después hay un bucle que constantemente está leyendo los datos que llegan a través del UART y los muestra una velocidad debajo de la otra en la pantalla.

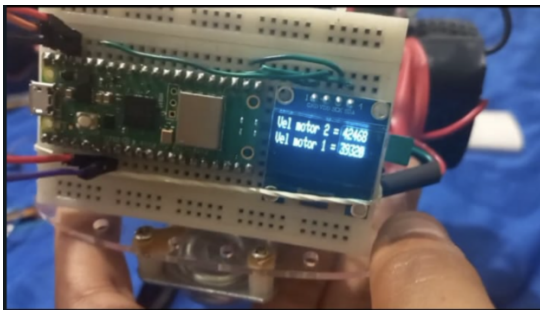

Figure 10: Pantalla OLED con las velocidades como números

En busca de otro modo de representar los datos en la pantalla se optó por representar los datos en forma de barras con este código:

```
def mostrar_barra(velocidad):
    # Limpiar la pantalla
    display.fill(0)

    # Calcular la longitud de la barra basada
        en la velocidad

    longitud_barra = int((velocidad / 120) *
        32)

    # Dibujar la barra vertical
    for y in range(32 - longitud_barra, 32):
        for x in range(10, 20):
            display.pixel(x, y, 1)

    # Actualizar la pantalla
    display.show()
```

Listing 21: Función que muestra las velocidades como barras

Esta función recibe la velocidad como un parámetro y se encarga de mostrar una barra de tamaño proporcional a dicha velocidad, el resultado fue:

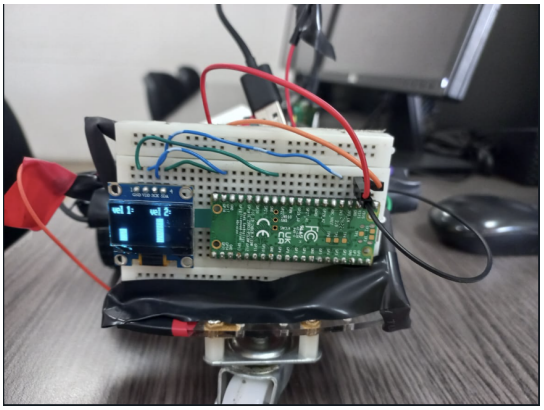

Figure 11: Pantalla OLED con las velocidades como barras

C. Consumo de energía:

La pila usada en la alimentacion de los motores del carro y el puente H, con la ayuda de un osciloscopio de mano se realizo la medicon de la corriente que esta carga exigia:

Figure 12: Corriente exigida por la carga

Hallando la potencia electrica consumida por la carga tenemos:

$$P = V * I = 7,2V * 198,53mA = 1,43W \quad (1)$$

Como la enerngia consumida se mide en vatio por hora, suponiendo un uso continuo de 2 horas tenemos:

$$E_C = P * t = 1,43W * 2h = 2,86Wh \quad (2)$$

VI. Conclusiones

Del proyecto se sacaron varias conclusiones la primera es que a través del uso del perceptrón se mejoró el rendimiento del carro, ya que con este el tiempo que toma el carro dando la vuelta disminuyó de un 1 minuto a 32 s.

También se consiguió utilizar distintos tipos de comunicaciones como la UART, I^2C y a través de un web server, para manejar diversos periféricos como la cámara y la pantalla OLED.

La cámara OV7670 resultó esencial para que el carro pudiera detectar y seguir la línea, ajustando su trayectoria en tiempo real con un algoritmo de cálculo de errores. Aunque nos enfrentamos a desafíos, como la sensibilidad de la cámara al brillo del ambiente, logramos que funcionara adecuadamente después de ajustar su posición y calibrarla correctamente. Esta experiencia nos enseñó mucho sobre cómo adaptarnos a diferentes condiciones, lo que nos ayudó a alcanzar los objetivos del proyecto.

Referencias

[1] Orionitalia.com . [En línea]. Disponible en: https://www.orionitalia.com/es/aplicaciones/industriales/que-es-un-motor-electrico. [Consultado: 28-feb-2024]. Example of scientific journal paper:

[2] Researchgate.net . [En línea]. Disponible en: https://www.researchgate.net/profile/Marino-Pernia/publication/235752021ConceptosBasicosdeMaquinas decorrientecontinua/links/0912f5131e8e23bfa1000000/Conceptos-Basicos-de-Maquinas-de-corriente-continua.pdf [Consultado: 28-feb-2024].