

Proyecto Final

1st Karen Tatiana Leon Olaya

20202005152

Bogotá, Colombia

Ktleono@udistrital.edu.co

Abstract—Este artículo detalla el desarrollo de un carro electrónico autónomo, que inicialmente en la primera fase, funcionaba como un carro a control remoto utilizando dos Raspberry Pi. La primera Raspberry Pi estaba equipada con CircuitPython para manejar la cámara OV7670, mientras que la segunda utilizaba MicroPython para gestionar el movimiento del carro. La segunda fase del proyecto convirtió el carro en un seguidor de línea utilizando la cámara como sensor. En el siguiente artículo se detalla y explica el desarrollo de cada fase del carro, mostrando los resultados obtenidos con el proyecto.

Index Terms—Raspberry Pi, AWR, cámara OV7670, MicroPython, CircuitPython, Seguidor de línea y red neuronal.

I. INTRODUCCIÓN

El desarrollo de vehículos autónomos ha avanzado significativamente en los últimos años, impulsado por mejoras en tecnologías de sensores, inteligencia artificial y sistemas embebidos. Estos vehículos, capaces de navegar sin intervención humana, tienen aplicaciones potenciales en diversas áreas, desde el transporte y la logística hasta la exploración y la vigilancia.

El proyecto presentado en este artículo ilustra un enfoque práctico, hacia la construcción de un carro electrónico autónomo, abarcando desde conceptos básicos de control remoto hasta técnicas avanzadas de inteligencia artificial. A lo largo de las tres fases del desarrollo, se utilizaron herramientas como CircuitPython y MicroPython, así como algoritmos de visión por computadora y aprendizaje por refuerzo, proporcionando una visión integral del proceso de desarrollo y las innovaciones tecnológicas implementadas. Este artículo detalla cada una de estas fases y presenta los resultados obtenidos a lo largo del semestre, destacando los retos superados y las lecciones aprendidas en el camino hacia la autonomía vehicular.

II. MARCO TEÓRICO

A. Definición y Funcionamiento

1) *Funcionamiento de la Raspberry Pi Pico y Raspberry Pi Pico W:*

- **La Raspberry Pi Pico** es una microcontroladora basada en el chip RP2040, diseñado por la Fundación Raspberry Pi. Este chip incluye un procesador dual-core ARM Cortex-M0+ con una frecuen-

cia de hasta 133 MHz, 264 KB de SRAM y 2 MB de memoria flash QSPI. Su arquitectura permite ejecutar código de manera eficiente para aplicaciones de tiempo real. La Pico cuenta con una variedad de interfaces periféricas, tales como UART, SPI, I2C, PWM, ADC y GPIOs, lo que la hace altamente versátil para proyectos de electrónica y desarrollo embebido. Además, se puede programar utilizando C/C++ y MicroPython, proporcionando flexibilidad en el desarrollo de software.

- **La Raspberry Pi Pico W** extiende las capacidades de la Pico original al incluir conectividad inalámbrica a través de un chip de Wi-Fi 802.11n (basado en el módulo CYW43439 de Infineon). Este módulo permite la conectividad a redes Wi-Fi, facilitando la creación de aplicaciones de IoT (Internet de las Cosas). Al igual que la Pico, la Pico W mantiene la misma configuración de hardware y puertos, garantizando compatibilidad y facilidad de uso en proyectos que requieren conectividad inalámbrica. Ambas placas se destacan por su bajo costo y alto rendimiento, haciéndolas ideales para una amplia gama de aplicaciones, desde el aprendizaje de la programación hasta la implementación de sistemas embebidos complejos.

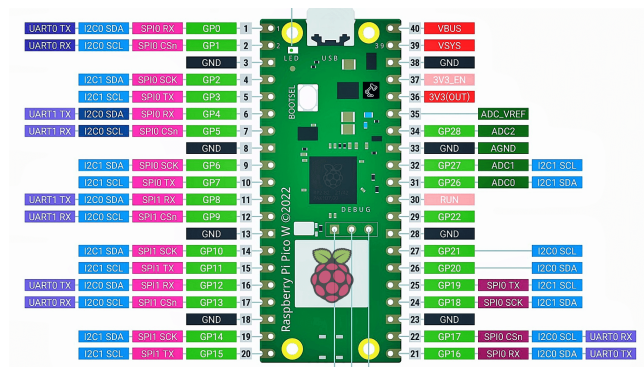


Fig. 1. Descripción de Pines Raspberry Pi Pico.



Fig. 2. Raspberry Pi Pico W.

2) Programación en CircuitPython y MicroPython:

- **CircuitPython:** CircuitPython es una variante del lenguaje de programación Python, diseñada específicamente para microcontroladores y aplicaciones de electrónica. Es mantenida por Adafruit y permite una fácil interacción con hardware y sensores, como la cámara OV7670 utilizada en este proyecto. CircuitPython se caracteriza por su simplicidad y facilidad de uso, lo que lo hace adecuado para proyectos educativos y de prototipado rápido.
- **MicroPython:** Similar a CircuitPython, MicroPython es una implementación del lenguaje Python optimizada para microcontroladores. Proporciona una plataforma eficiente y compacta para desarrollar aplicaciones de control y automatización. En este proyecto, MicroPython se utilizó en la Raspberry Pi Pico W para gestionar el movimiento del carro y la comunicación con otros dispositivos.

3) **Camara OV7670:** La **cámara OV7670** Es un módulo de sensor de imagen CMOS de baja potencia, diseñado para capturar imágenes a resolución VGA (640x480) a 30 fps. Este módulo es ampliamente utilizado en proyectos de electrónica y sistemas embebidos debido a su bajo costo y facilidad de integración. La OV7670 cuenta con una interfaz SCCB (Serial Camera Control Bus), similar a I2C, para la configuración de sus registros internos, y una interfaz de datos paralela de 8 bits para la salida de las imágenes capturadas. El sensor puede ajustar parámetros de imagen como exposición, balance de blancos, saturación y contraste a través de sus registros. Además, soporta diversas resoluciones y formatos de salida, incluyendo RGB565, YUV422 y GRB422. La simplicidad de su diseño y la flexibilidad en la configuración hacen de la OV7670 una opción popular para aplicaciones de visión por computadora, sistemas de vigilancia y proyectos educativos.



Fig. 3. Camara OV7670

III. EXPLICACION FASES

IV. CÓDIGOS

A. Código en Python de la primera fase

Listing 1. CODE Camara OV7670

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

from adafruit_ov7670 import (
    OV7670,
    OV7670_SIZE_DIV16,
    OV7670_COLOR_YUV,
    OV7670_TEST_PATTERN_COLOR_BAR_FADE,
)

cam_bus = busio.I2C(board.GP21, board.GP20)
cam = OV7670(
    cam_bus,
    data_pins=[
        board.GP0,
        board.GP1,
        board.GP2,
        board.GP3,
        board.GP4,
        board.GP5,
        board.GP6,
        board.GP7,
    ],
    clock=board.GP8,
    vsync=board.GP13,
    href=board.GP12,
    mclk=board.GP9,
    shutdown=board.GP15,
    reset=board.GP14,
)

cam.size = OV7670_SIZE_DIV16
cam.colormap = OV7670_COLOR_YUV
cam.flip_y = True

print(cam.width, cam.height)

# Capturar la primera imagen
first_image = bytearray(2 * cam.width * cam.height)
cam.capture(first_image)
print("Primera imagen capturada")

buf = bytearray(2 * cam.width * cam.height)
width = cam.width
# Definir las posiciones de referencia del primer y último cero
ref_first_zero_pos = 15
ref_last_zero_pos = 25
total_error_first = 0
total_error_last = 0
num_lines = 0
direction = None
straight_count = 0 # Contador de desviaciones rectas
diagonal_count = 0 # Contador de desviaciones diagonales

while True:
    # Capturar la imagen actual
    cam.capture(buf)

    # Inicializar variables para el calculo del error promedio
    total_error_first = 0
    total_error_last = 0
    num_lines = 0
    straight_count = 0
    diagonal_count = 0

    # Lista para almacenar las posiciones de los ceros en el primer y último
    # cero
    first_zero_positions = []
    last_zero_positions = []

    # Mostrar solo las líneas del 10 al 20 de la imagen actual
    for j in range(10, 21):
        # Inicializar una cadena vacía para representar la línea binarizada
        binary_line = ""

        # Iterar sobre los píxeles en la línea actual
        for i in range(cam.width):
            intensity = buf[2 * (width * j + i)]

            # Si la intensidad es menor o igual a 100, considerar el píxel
            # como negro (0),
            # de lo contrario, considerarlo como blanco (1)
            pixel_value = 0 if intensity <= 100 else 1

            # Agregar el valor del píxel binarizado a la cadena
            binary_line += str(pixel_value)

        # Encontrar la posición del primer cero
        zero_pos = binary_line.find('0')
        # Encontrar la posición del último cero
        last_zero_pos = binary_line.rfind('0')

        # Actualizar las listas de posiciones de ceros
        first_zero_positions.append(zero_pos)
        last_zero_positions.append(last_zero_pos)
```

```

# Calcular el error porcentual para el primer cero
if zero_pos == -1:
    percent_error_first = 100
else:
    percent_error_first = abs((zero_pos - ref_first_zero_pos) * 100
        / ref_first_zero_pos)

# Calcular el error porcentual para el último cero
if last_zero_pos == -1:
    percent_error_last = 100
else:
    percent_error_last = abs((last_zero_pos - ref_last_zero_pos) *
        100 / ref_last_zero_pos)

# Si el porcentaje es mayor que 100, ajustarlo a 100
percent_error_first = min(percent_error_first, 100)
percent_error_last = min(percent_error_last, 100)

# Actualizar el total de errores y el número de líneas
total_error_first += percent_error_first
total_error_last += percent_error_last
num_lines += 1

# Determinar si la desviación es recta
if percent_error_first == percent_error_last:
    straight_count += 1
else:
    diagonal_count += 1

# Imprimir la línea binarizada y el error porcentual
print(f"Línea {j}: {binary_line}, Error primer cero: {
    percent_error_first:.2f}%, Error último cero: {
    percent_error_last:.2f}%")

# Calcular el error promedio para el primer y último cero
avg_error_first = total_error_first / num_lines
avg_error_last = total_error_last / num_lines

# Determinar si la desviación es recta o diagonal
if straight_count >= 5 or any(first_zero_positions.count(pos) >= 5 for
    pos in first_zero_positions):
    # Si más de la mitad de las líneas tienen un patrón consistente o
    # alguna posición de ceros se repite mucho,
    # considerar la desviación como recta
    direction = "recta"
else:
    direction = "diagonal"

# Determinar la dirección de la desviación
# Si la desviación es recta, determinar la dirección basada en la
# posición del primer cero
if zero_pos < ref_first_zero_pos:
    deviation_direction = "izquierda"
else:
    # Si la desviación es diagonal, no se puede determinar una dirección
    # específica
    deviation_direction = "no se puede determinar"
if zero_pos > ref_first_zero_pos:
    deviation_direction = "derecha"
else:
    # Si la desviación es diagonal, no se puede determinar una dirección
    # específica
    deviation_direction = "no se puede determinar"
print(zero_pos)
# Imprimir resultados
print(f"Error promedio primer cero: {avg_error_first:.2f}%")
print(f"Error promedio último cero: {avg_error_last:.2f}%")
print(f"Desviación: {direction} con desviación hacia {
    deviation_direction}")
time.sleep(2)

```

Listing 2. CODE Control Remoto

```

import network
import socket
from time import sleep
import machine
from machine import Pin, PWM

ssid = 'Snow'
password = 'A2720528'

# Configurar PWM para controlar la velocidad de la llanta A y B
pwm_A = PWM(Pin(17)) # Pin GPIO 17 para controlar la velocidad de la llanta
    A
pwm_A.freq(1000) # Frecuencia de PWM
pwm_B = PWM(Pin(16)) # Pin GPIO 16 para controlar la velocidad de la llanta
    B
pwm_B.freq(1000) # Frecuencia de PWM

velocidad_A = 65535 # Valor de ciclo de trabajo PWM inicial para llanta A (
    rango: 0-65535)
velocidad_B = 65535 # Valor de ciclo de trabajo PWM inicial para llanta B (
    rango: 0-65535)
#pwm_A.duty_u16(velocidad_A) # Establecer el ciclo de trabajo PWM inicial
    para llanta A
#pwm_B.duty_u16(velocidad_B) # Establecer el ciclo de trabajo PWM inicial
    para llanta B

def adelante():
    Motor_A.Adelante.value(1)
    Motor_B.Adelante.value(1)
    Motor_A.Atras.value(0)
    Motor_B.Atras.value(0)

```

```

pwm_A.duty_u16(60000) # Reducir la velocidad de la llanta A cuando se
    mueve hacia la izquierda
pwm_B.duty_u16(60000) # Aumentar la velocidad de la llanta B cuando se
    mueve hacia la izquierda

def atras():
    Motor_A.Adelante.value(0)
    Motor_B.Adelante.value(0)
    Motor_A.Atras.value(1)
    Motor_B.Atras.value(1)
    pwm_A.duty_u16(60000) # Establecer el ciclo de trabajo PWM inicial para
        llanta A
    pwm_B.duty_u16(60000) # Establecer el ciclo de trabajo PWM inicial para
        llanta B

def detener():
    Motor_A.Adelante.value(0)
    Motor_B.Adelante.value(0)
    Motor_A.Atras.value(0)
    Motor_B.Atras.value(0)
    pwm_A.duty_u16(0) # Detener llanta A
    pwm_B.duty_u16(0) # Detener llanta B

def izquierda():
    Motor_A.Adelante.value(0)
    Motor_B.Adelante.value(1)
    Motor_A.Atras.value(0)
    Motor_B.Atras.value(0)
    pwm_A.duty_u16(0) # Reducir la velocidad de la llanta A cuando se mueve
        hacia la izquierda
    pwm_B.duty_u16(60000) # Aumentar la velocidad de la llanta B cuando se
        mueve hacia la izquierda

def derecha():
    Motor_A.Adelante.value(1)
    Motor_B.Adelante.value(0)
    Motor_A.Atras.value(0)
    Motor_B.Atras.value(0)
    pwm_A.duty_u16(60000) # Aumentar la velocidad de la llanta A cuando se
        mueve hacia la derecha
    pwm_B.duty_u16(0) # Reducir la velocidad de la llanta B cuando se mueve
        hacia la derecha

detener()

def conectar():
    red = network.WLAN(network.STA_IF)
    red.active(True)
    red.connect(ssid, password)
    while red.isconnected() == False:
        print('Conectando ...')
        sleep(1)
    ip = red.ifconfig()[0]
    print(f'Conectado con IP: {ip}')
    return ip

def open_socket(ip):
    address = (ip, 80)
    connection = socket.socket()
    connection.bind(address)
    connection.listen(1)
    return connection

def pagina_web():
    html = """
    <DOCTYPE html>
    <html>
    <head>
    <style>
    <button {
        touch-action: manipulation;
    }
    </style>
    <script>
        function enviarComando(comando) {
            var xhttp = new XMLHttpRequest();
            xhttp.open("GET", "/" + comando, true);
            xhttp.send();
        }

        function presionado(comando) {
            enviarComando(comando);
            setTimeout(function(){ enviarComando('detener'); },
                1000);
        }
    </script>
    </head>
    <body>
    <center>
    <button id="adelante" ontouchstart="presionado('adelante')">
        Adelante </button>
    <table>
    <tr>
    <td><button id="izquierda" ontouchstart="presionado('izquierda')
        ">Izquierda </button></td>
    <td><button id="detener" disabled>Detener </button></td>
    <td><button id="derecha" ontouchstart="presionado('derecha')">
        Derecha </button></td>
    </tr></table>
    <button id="atras" ontouchstart="presionado('atras')">Atras </
        button>
    </body>
    </html>
    """
    return html

def serve(connection):
    while True:

```

```

        cliente = connection.accept()[0]
        petition = cliente.recv(1024)
        petition = str(petition)
        try:
            petition = petition.split()[1]
        except IndexError:
            pass
        if petition == '/adelante':
            adelante()
            sleep(2)
            detener()
        elif petition == '/izquierda':
            izquierda()
            sleep(2)
            detener()
        elif petition == '/derecha':
            derecha()
            sleep(2)
            detener()
        elif petition == '/atras':
            atras()
            sleep(2)
            detener()
        cliente.send(pagina_web())
        cliente.close()

try:
    ip = conectar()
    connection = open_socket(ip)
    serve(connection)
except KeyboardInterrupt:
    machine.reset()

```

B. Código en Python de la segunda fase

Listing 3. CODE Camara Seguidor De linea

```

import sys
import time
import digitalio
import busio
import board
from adafruit_ov7670 import (
    OV7670,
    OV7670_SIZE_DIV16,
    OV7670_COLOR_YUV,
)

# Configurar el UART
uart = busio.UART(board.GP16, board.GP17, baudrate=115200)

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

# Función para enviar texto a través de UART
def enviar_texto(texto):
    uart.write(texto.encode('utf-8'))

# Configuración de la cámara OV7670
cam_bus = busio.I2C(board.GP21, board.GP20)

cam = OV7670(
    cam_bus,
    data_pins=[
        board.GP0,
        board.GP1,
        board.GP2,
        board.GP3,
        board.GP4,
        board.GP5,
        board.GP6,
        board.GP7,
    ],
    clock=board.GP8,
    vsync=board.GP13,
    href=board.GP12,
    mclk=board.GP9,
    shutdown=board.GP15,
    reset=board.GP14,
)
cam.size = OV7670_SIZE_DIV16
cam.colourspace = OV7670_COLOR_YUV
cam.flip_y = True

print(cam.width, cam.height)

# Capturar la primera imagen para inicializar la cámara
first_image = bytearray(2 * cam.width * cam.height)
cam.capture(first_image)
print("Primera imagen capturada")

# Buffer para capturar la imagen
buf = bytearray(2 * cam.width * cam.height)

width = cam.width
height = cam.height

while True:
    # Capturar la imagen actual
    cam.capture(buf)

    # Inicializar una cadena vacía para representar la línea binarizada

```

```

binary_line = ""

# Iterar sobre los píxeles en la última línea
for i in range(cam.width):
    intensity = buf[2 * (width * (height - 1) + i)]

    # Si la intensidad es menor o igual a 100, considerar el píxel como
    # negro (0).
    # de lo contrario, considerarlo como blanco (1)
    pixel_value = 0 if intensity <= 100 else 1

    # Agregar el valor del píxel binarizado a la cadena
    binary_line += str(pixel_value)

print(binary_line)

# Opcional: enviar la línea binarizada a través de UART
enviar_texto(binary_line)

# Dormir por un breve periodo de tiempo para reducir la velocidad del
# bucle si es necesario
time.sleep(0.1)

```

Listing 4. CODE Seguidor De linea

```

from machine import Pin, PWM, UART
import utime

# Configurar la conexión UART (TX: Pin 0, RX: Pin 1)
uart = UART(0, baudrate=115200, tx=Pin(0), rx=Pin(1))

# Configurar los pines de los motores con PWM para control de velocidad
Motor_A_Adelante = PWM(Pin(18))
Motor_A_Adelante.freq(1000)
Motor_A_Adelante.duty_u16(0)

Motor_A_Atras = PWM(Pin(19))
Motor_A_Atras.freq(1000)
Motor_A_Atras.duty_u16(0)

Motor_B_Adelante = PWM(Pin(20))
Motor_B_Adelante.freq(1000)
Motor_B_Adelante.duty_u16(0)

Motor_B_Atras = PWM(Pin(21))
Motor_B_Atras.freq(1000)
Motor_B_Atras.duty_u16(0)

# Configurar el pin 16 como salida
led = Pin("LED", Pin.OUT)

buffer = "" # Inicializar el buffer vacío
direccion_anterior = None # Variable para almacenar la dirección anterior

def interpretar_direccion(renglon):
    global direccion_anterior

    if all(c == 'l' for c in renglon):
        if direccion_anterior == 'izquierda':
            return 'derecha' # Si es todo unos y la dirección anterior era
            # izquierda, girar a la derecha
        elif direccion_anterior == 'derecha':
            return 'izquierda' # Si es todo unos y la dirección anterior
            # era derecha, girar a la izquierda

        elif direccion_anterior == 'adelante':
            return 'atras' # Si es todo unos y la dirección anterior era
            # derecha, girar a la izquierda

# Verificar si hay ceros y unos intercalados

# Contar los ceros y unos en el renglón
num_ceros = renglon.count('0')
num_unos = renglon.count('1')

# Si hay más ceros que unos, tomar la dirección como "adelante"
if num_ceros > num_unos:
    return 'adelante'

# Verificar si el renglón consiste en todos unos

# Verificar el primer y último carácter del renglón
primer_caracter = renglon[0]
ultimo_caracter = renglon[-1]

# Determinar la dirección basándose en los ceros en el borde
if primer_caracter == '0':
    direccion = 'izquierda' # Girar a la derecha si hay un cero al
    # inicio
elif ultimo_caracter == '0':
    direccion = 'derecha' # Girar a la izquierda si hay un cero al
    # final
else:
    direccion = 'adelante' # Avanzar si no hay ceros en el borde

# Actualizar la dirección anterior
direccion_anterior = direccion

return direccion

# Función para controlar los motores según la dirección
def controlar_motores(direccion):

```

```

if direccion == 'detener':
    Motor_A_Adelante.duty_u16(0)
    Motor_A_Atras.duty_u16(0)
    Motor_B_Adelante.duty_u16(0)
    Motor_B_Atras.duty_u16(0)
elif direccion == 'adelante':
    Motor_A_Adelante.duty_u16(50000) # 50% del ciclo de trabajo (mitad
    de la velocidad)
    Motor_A_Atras.duty_u16(0)
    Motor_B_Adelante.duty_u16(65000)
    Motor_B_Atras.duty_u16(0)
elif direccion == 'derecha':
    Motor_A_Adelante.duty_u16(50000)
    Motor_A_Atras.duty_u16(0)
    Motor_B_Adelante.duty_u16(0)
    Motor_B_Atras.duty_u16(0)
elif direccion == 'izquierda':
    Motor_A_Adelante.duty_u16(0)
    Motor_A_Atras.duty_u16(0)
    Motor_B_Adelante.duty_u16(65000)
    Motor_B_Atras.duty_u16(65000)
elif direccion == 'atras':
    Motor_A_Adelante.duty_u16(0) # 50% del ciclo de trabajo (mitad de
    la velocidad)
    Motor_A_Atras.duty_u16(50000)
    Motor_B_Adelante.duty_u16(0)
    Motor_B_Atras.duty_u16(65000)

while True:
    try:
        if uart.any():
            datos_recibidos = uart.read(40) # Leer hasta 40 caracteres
            if datos_recibidos:
                try:
                    texto = datos_recibidos.decode('utf-8').strip()
                    buffer = texto

                    # Imprimir el renglón recibido
                    print("Renglón recibido:", buffer)

                    # Interpretar la dirección
                    direccion = interpretar_direccion(buffer)
                    print("Dirección tomada:", direccion)

                    # Controlar los motores
                    controlar_motores(direccion)

                    # Encender el LED al recibir datos
                    led.toggle()
                except UnicodeError:
                    print("Error al decodificar datos recibidos")
                    direccion = 'detener'
                    controlar_motores(direccion)

    except KeyboardInterrupt:
        break

# Detener todos los motores al salir
Motor_A_Adelante.duty_u16(0)
Motor_A_Atras.duty_u16(0)
Motor_B_Adelante.duty_u16(0)
Motor_B_Atras.duty_u16(0)

# Cerrar la conexión UART al salir
uart.deinit()

```

V. COMPONENTES Y PRECIOS

- Raspberry Pi Pico W: Es la unidad central de control del carro, manejando las operaciones y comunicaciones del sistema.
- Cámara OV7670: Proporciona capacidades de visión al carro, esencial para la navegación y el seguimiento de líneas.
- Chasis para motor: Proporciona la estructura física y los motores necesarios para el movimiento del carro.
- Batería Recargable: Suministra la energía necesaria para operar todos los componentes electrónicos.
- Protoboard: Facilita las conexiones y la configuración del circuito sin necesidad de soldaduras.
- Puente H: Controla la dirección y velocidad de los motores del carro.
- Jumpers: Son necesarios para realizar las conexiones entre los diferentes componentes electrónicos.



Elemento	Foto	Precio (COP)	Cantidad
Raspberry Pi Pico W		35.700	2
Cámara OV7670		20.900	1
chasis Para motor		100,000	1
Batería Recar-gable		20,000	1
Protoboard		12,000	1
Puente H		11,000	1
Jumpers (Hembra-macho/ macho - macho)		8,000	20

TABLE I
ELEMENTOS UTILIZADOS EN EL PROYECTO DEL CARRO ELECTRÓNICO

VI. RESULTADOS

A. Implementación del diseño

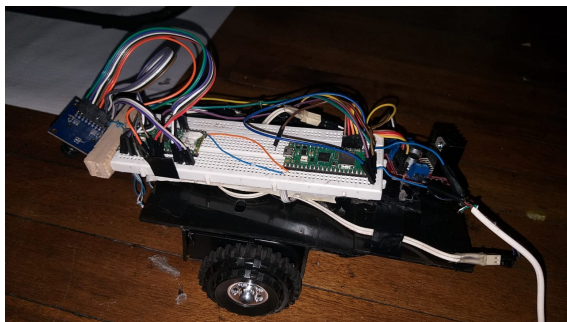
1) *Uso y manejo de la Camara OV7670:* Para el manejo de la camara se procedio a realizar pruebas y entender el funcionamiento de la misma, explorando en las funciones y usos de este elemento:

Para conectar la Raspberry Pi Pico W a una red Wi-Fi, primero necesitamos importar las librerías necesarias y configurar los parámetros de la red.

Una vez conectada la Raspberry Pi Pico W a una red Wi-Fi, podemos enviar datos a un servidor web. A continuación se muestra un ejemplo de cómo enviar una solicitud HTTP GET a un servidor.

```
import urequests
url = 'http://ejemplo.com/datos'
response = urequests.get(url)
print('Código_de_estado:', response.status_code)
print('Contenido:', response.text)
```

Como se observa en el ejemplo se asigno a el "color" observado un numero con el cual se podia controlar la direccion de acuerdo a la cantidad de 1 o 0 que estuvieran presentes en la vista de la camara donde 0 es la línea.



2) *Control remoto:* Para el control remoto se realizo una pagina para controlar y predecir las señales y funcionamiento del carro, aprovechando las virtudes de la Raspberry Pi Pico W y su funcionalidad wi-fi. Antes de comenzar, asegúrate de tener instaladas las herramientas necesarias:

- 3) *Seguidor de línea:* La fase del seguidor de línea implemento todas las fases anteriores en busca del objetivo del tiempo y la precision en el desplazamiento:

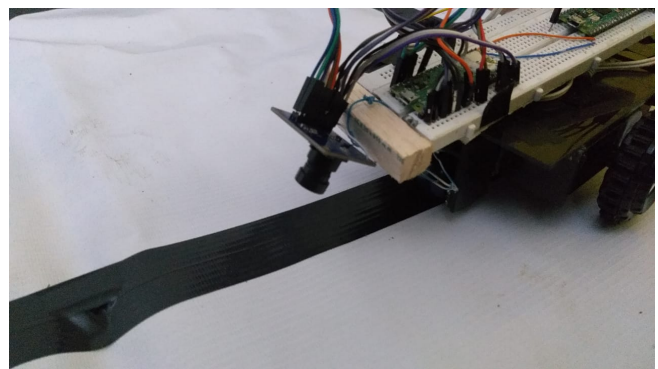


Fig. 8. Seguidor de línea

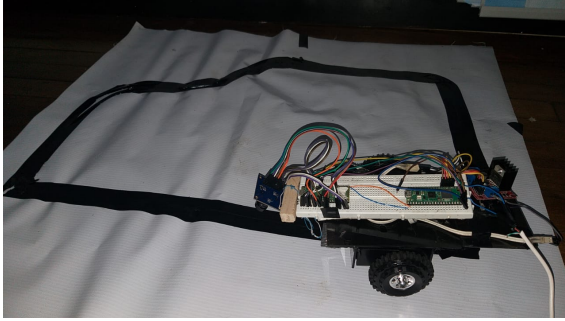


Fig. 9. carro seguidor de línea sobre la pista

B. Resultados obtenidos

Del trabajo en el semestre, se obtuvieron los siguientes resultados:

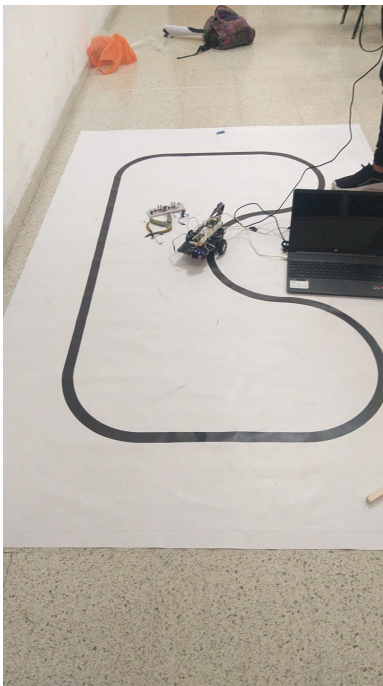


Fig. 10. Carro en funcionamiento con la pista.

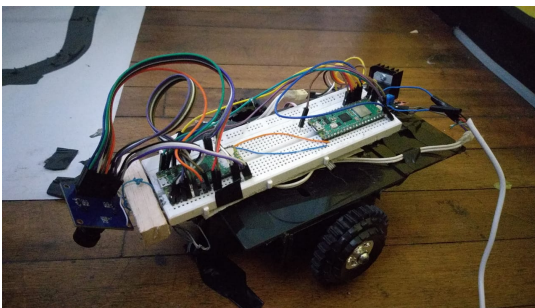


Fig. 11. Prototipo final del carro.

C. Errores Encontrados

Durante la fase de control remoto, se intentó implementar una página web con control por joystick, pero surgieron varios problemas. En primer lugar, se observó un problema de velocidad y de direcciones, ya que la división entre una dirección y otra era muy estrecha, lo que provocaba conflictos sobre qué dirección tomaría el carro. Esto resultaba en choques y comportamientos erráticos del vehículo.

Además, en esta fase no se pudo implementar la comunicación UART, por lo que la información tuvo que ser enviada mediante PWM, lo que retrasó el envío de los datos. Este retraso se debió a la necesidad de codificar la información en la Raspberry Pi Pico con CircuitPython, enviarla a otra Raspberry Pi Pico con MicroPython y luego decodificarla. Este proceso introdujo latencias adicionales que afectaron el rendimiento general del sistema.

En la fase de seguimiento de líneas, se encontraron otros problemas. A veces, la cámara detectaba su propia sombra como si fuese una línea y el carro seguía esta sombra en lugar de la línea real. Se intentó solucionar este problema implementando LEDs, pero estos no resultaron ser efectivos.

En esta fase, sin embargo, se logró implementar la comunicación UART. No obstante, fue necesario reducir el número de renglones de datos enviados. En la fase de control remoto se utilizaban 10 renglones, mientras que en esta fase se redujo a solo uno para que los datos pudieran ser enviados más rápidamente a una velocidad de 115200 baudios. Esto permitió que el carro tuviera tiempo suficiente para tomar decisiones rápidas, ya que enviar los 10 renglones de datos consecutivamente resultaba en una toma de decisiones demasiado lenta.

• Fase de control remoto:

- Problemas de velocidad y de direcciones debido a la estrecha división entre ellas, causando choques en la dirección del carro.
- No se pudo implementar UART; se utilizó PWM, lo cual retrasó el envío de datos por la codificación y decodificación entre Raspberry Pi Pico con CircuitPython y MicroPython.

• Fase de seguimiento de líneas:

- La cámara a veces detectaba su propia sombra como línea.
- Intento fallido de solucionar el problema con LEDs.
- Implementación de UART con reducción de renglones de datos enviados de 10 a 1, mejorando la velocidad de decisión del carro a 115200 baudios.

VII. CONCLUSIONES

- 1) El desarrollo de un carro autónomo utilizando la Raspberry Pi Pico W demostró ser una tarea factible y educativa. A lo largo del proyecto, se integraron varias tecnologías como CircuitPython, MicroPython, lo que permitió a los estudiantes obtener una comprensión profunda sobre el control de hardware y la programación embebida. Esta experiencia práctica no solo mejoró las habilidades técnicas, sino que también les proporcionó un valioso conocimiento sobre los desafíos y soluciones en la implementación de vehículos.
- 2) El uso de la cámara OV7670 como sensor de visión fue crucial para el éxito del proyecto. La integración de este componente permitió al carro detectar y seguir líneas en la pista, facilitando así la implementación del control autónomo. La experiencia adquirida en el manejo de sensores de visión y su aplicación práctica en el control de vehículos proporciona una base sólida para futuros proyectos en el campo de la visión por computadora y la robótica autónoma.
- 3) El proyecto presentó una serie de desafíos técnicos y logísticos que fueron abordados con diversas soluciones. A continuación, se presentan las conclusiones más relevantes del proyecto:
 - a) **Adaptabilidad y Flexibilidad:** La capacidad de adaptación fue crucial para enfrentar los problemas imprevistos que surgieron durante el desarrollo. La transición de la comunicación UART a PWM en la fase de control remoto, y el ajuste del número de renglones en la fase de seguimiento de líneas, son ejemplos de cómo se implementaron soluciones alternativas para asegurar el funcionamiento del sistema.
 - b) **Desempeño del Sistema:** La implementación de una página web con control por joystick permitió una interacción remota, aunque los problemas de velocidad y precisión en la dirección revelaron la necesidad de mejoras en la interfaz de usuario y en la gestión de comandos. La experiencia obtenida en esta fase proporcionó una valiosa comprensión sobre las limitaciones y capacidades de los dispositivos utilizados.
 - c) **Optimización de Comunicación:** La optimización de la comunicación fue un aspecto crítico en ambas fases del proyecto. En la fase de control remoto, la comunicación por PWM, aunque subóptima, permitió continuar con las pruebas. En la fase de seguimiento de líneas, la implementación exitosa de UART y

la reducción de la cantidad de datos enviados mejoraron significativamente la velocidad de respuesta del sistema.

- d) **Problemas de Detección:** La detección de líneas utilizando la cámara presentó desafíos inesperados, como la interpretación errónea de sombras como líneas. Este problema destacó la importancia de las condiciones de iluminación y de un procesamiento de imagen robusto para sistemas de seguimiento de líneas. Aunque la implementación de LEDs no resolvió completamente el problema, proporcionó insights para futuras mejoras en el sistema de detección.
- e) **Lecciones Aprendidas:** Cada problema enfrentado y solucionado durante el proyecto aportó lecciones valiosas. La necesidad de una mejor calibración y configuración de los sensores, la importancia de una comunicación eficiente y la adaptación rápida a los cambios imprevistos son aprendizajes que mejorarán futuros desarrollos.
- f) **Potencial de Mejora:** A pesar de los desafíos, el proyecto demostró un gran potencial de mejora. Las futuras iteraciones pueden beneficiarse de una mejor integración de hardware y software, optimizaciones en la detección y procesamiento de imágenes, y mejoras en la interfaz de usuario para el control remoto.

REFERENCES

- [1] Clase Magistral impartida por el docente Gerardo Muñoz.
- [2] Adafruit, "CircuitPython Overview," Available: <https://circuitpython.org/>
- [3] MicroPython, "MicroPython Documentation," Available: <https://docs.micropython.org/>
- [4] OV7670 Camera Module Datasheet, Available: <https://www.arducam.com/camera-modules/ov7670/>