

Carro seguidor de linea

Edicson Rondon Carrillo
Cod. 20191005121
Universidad Distrital Francisco Jose de Caldas
Bogota D.C
erondonc@udistrital.edu.co

Omar Aturo castañeda Ospina
Cod. 20222005006
Universidad Distrital Francisco Jose de Caldas
Bogota D.C
oacastaneda@udistrital.edu.co

Abstract—Este informe presenta el desarrollo y la implementación de un automóvil robótico de seguimiento de líneas que utiliza una cámara OV7670 y una pantalla OLED SSD1306. El sistema está alimentado por un microcontrolador Raspberry Pi Pico W. Las imágenes capturadas se procesan para detectar la posición de la línea y, en base a estos datos, el automóvil ajusta su dirección y velocidad. Este documento incluye la instalación y configuración de los componentes de hardware, la implementación del software y la funcionalidad del algoritmo de seguimiento de línea.

Index Terms—circuitpython, ov7670 camera

I. INTRODUCCION

En los últimos años, la robótica ha ganado un importante impulso en diversos campos, incluida la automatización industrial, la electrónica de consumo y los proyectos educativos. Un aspecto importante que puede desarrollar la robótica de manera muy básica la capacidad de seguir una línea, lo cual sirve de base para tareas de navegación más complejas. Este proyecto tiene como objetivo construir un automóvil robótico simple que siga líneas utilizando una Raspberry Pi Pico W, una cámara OV7670 para detección de líneas y una pantalla OLED SSD1306 para retroalimentación de instrucciones en tiempo real. La implementación implica capturar imágenes, procesarlas para identificar la línea y controlar los motores del automóvil a partir de ello.

II. ELEMENTOS DE LA IMPLEMENTACIÓN

A. Microprocesador

La Raspberry Pi Pico W es una placa microcontroladora basada en el chip microcontrolador RP2040, diseñada por Raspberry Pi. Cuenta con dos núcleos ARM Cortex-M0+, 264 KB de SRAM y 2 MB de memoria flash integrada. La placa ofrece una variedad de opciones de E/S, incluidas GPIO, I2C, SPI y UART, lo que la hace adecuada para diversas aplicaciones integradas.

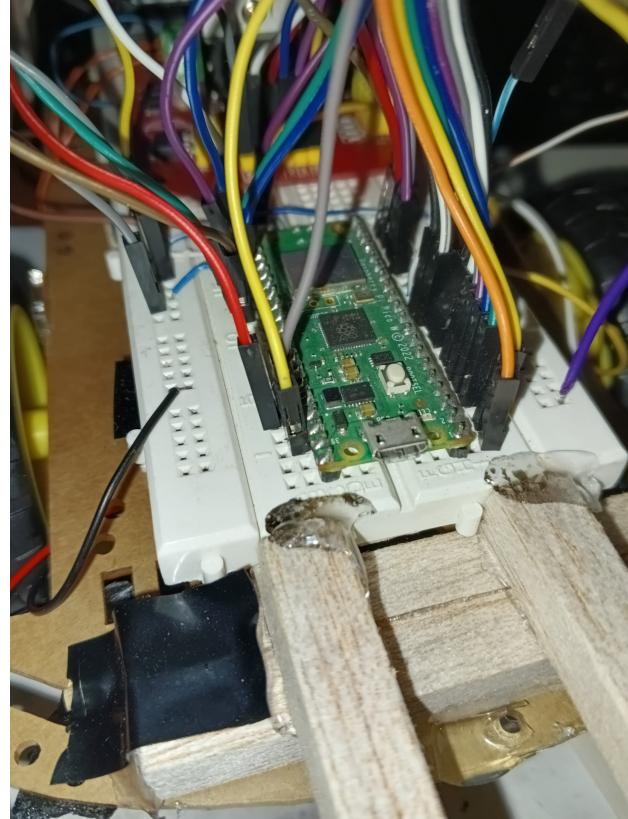


Fig. 1: Raspberry Pi Pico W

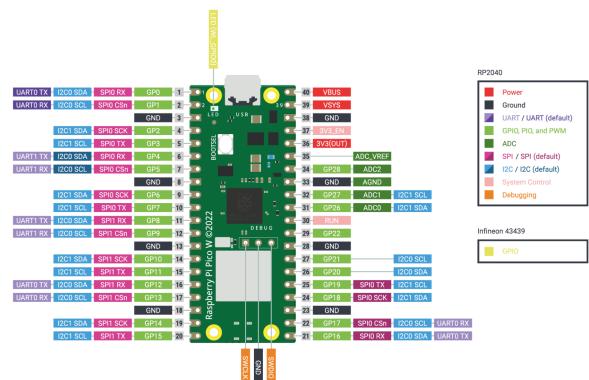


Fig. 2: pin out de la Raspberry Pi Pico W

B. Cámara

El OV7670 es un sensor de imagen de bajo costo capaz de capturar imágenes con resolución VGA de 640x480. Admite varios formatos de salida, incluidos RGB y YUV, y puede conectarse con microcontroladores a través de un bus paralelo. La cámara se utiliza ampliamente en proyectos educativos y de aficionados debido a su asequibilidad y facilidad de integración.

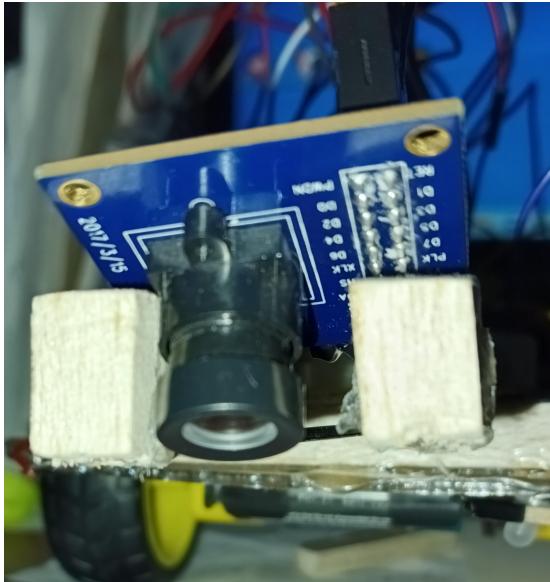


Fig. 3: Cámara OV7670

C. Pantalla de visualización

El SSD1306 es un controlador de pantalla OLED monocromático con una resolución de 128x64 píxeles. Se comunica con microcontroladores a través de interfaces I2C o SPI. Las pantallas OLED son conocidas por su alto contraste y bajo consumo de energía, lo que las hace ideales para aplicaciones que funcionan con baterías.

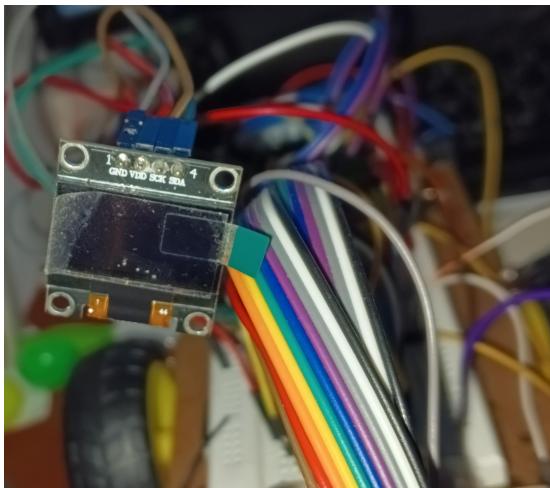


Fig. 4: Oled

D. L298N

El driver puente H L298N es el módulo más utilizado para manejar motores DC de hasta 2 amperios. El chip L298N internamente posee dos puentes H completos que permiten controlar 2 motores DC o un motor paso a paso bipolar/unipolar.

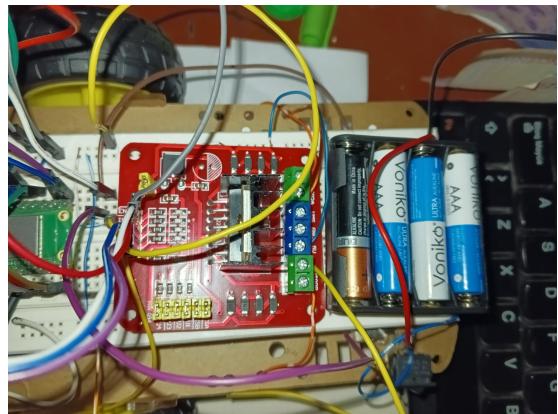


Fig. 5: Puente H

E. Chasis 2WD KIT Carro

Estructura especialmente diseñada para hacer el montaje de un carro electrónico y/o mecánico, cuenta con una serie de perforaciones sobre el chasis lo que facilita la instalación de cables y módulos que se requieran como un microcontrolador. Puede ser utilizado con funciones de rastreo, evasión de obstáculos, pruebas de distancia, velocidad y cualquier proyecto relacionado.

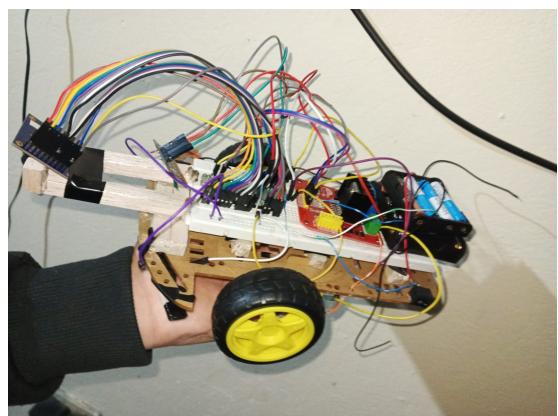


Fig. 6: Chasis

CONTENIDO DEL PAQUETE

- 1 x Chasis en acrílico
- 2 x Motores con caja reductora
- 2 x Llantas
- 2 x Encoder
- 1 x Rueda loca
- 1 x Porta baterías

Juego de tuercas y tornillos
Instrucciones para ensamble

Medidas: 15X21 cm Parte Ancha y 21 x 10 cm parte delgada

F. Arreglo LED

Dado el problema de la oscuridad en la lectura de la cámara, se planteo como posible solución la implementación de un arreglo de diodos LED en paralelo, esto con el fin de reducir la interferencia de sombras en la cámara.

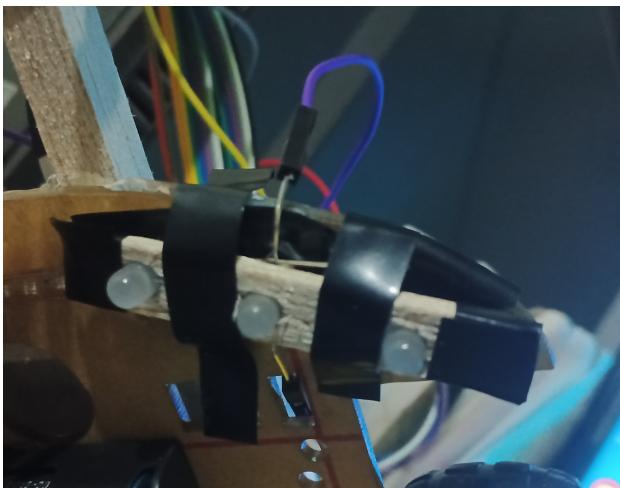


Fig. 7: Arreglo de diodos LED

G. Librerías

Varias bibliotecas facilitan la interfaz y el control de los componentes de hardware en este proyecto:

- "board" y "digitalio" para el control de GPIO.
- "busio" para la comunicación I2C.
- "pwmio" para generación de señal PWM.
- "displayio" y "adafruit-display-text.label" para manejar la pantalla OLED.
- "adafruit-displayio-ssd1306.SSD1306" para el controlador de pantalla OLED.
- "time" para los tiempos se espera, más conocidos como "sleep".
- "adafruit-ov7670.OV7670" para el controlador de la cámara.

III. IMPLEMENTACIÓN DEL CÓDIGO

A. Configuraciones

1) Configuración de los pines del carro: :

```
1 m11 = digitalio.DigitalInOut(board.GP20)
2 m11.direction = digitalio.Direction.
3     OUTPUT
4 m12 = digitalio.DigitalInOut(board.GP10)
```

```
5 m12.direction = digitalio.Direction.
6     OUTPUT
7 m21 = digitalio.DigitalInOut(board.GP22)
8 m21.direction = digitalio.Direction.
9     OUTPUT
10 m22 = digitalio.DigitalInOut(board.GP11)
11 m22.direction = digitalio.Direction.
12     OUTPUT
```

2) Configuración de la cámara: :

```
2 cam_bus = busio.I2C(board.GP19, board.
3     GP18)
4 cam = OV7670(
5     cam_bus,
6     data_pins=[
7         board.GP0,
8         board.GP1,
9         board.GP2,
10        board.GP3,
11        board.GP4,
12        board.GP5,
13        board.GP6,
14        board.GP7,
15     ],
16     clock=board.GP9,
17     vsync=board.GP13,
18     href=board.GP12,
19     mclk=board.GP8,
20     shutdown=board.GP15,
21     reset=board.GP14,
22 )
23
24 cam.size = OV7670_SIZE_DIV16
25 cam.colorspace = OV7670_COLOR_YUV
26 cam.flip_y = True
```

3) Configuración de PWM para controlar la velocidad del carro: :

```
1 pwm = pwmio.PWMOut(board.GP21, frequency
2     =100, duty_cycle=0)
```

4) Liberar recursos del display (Optional): :

```
1 displayio.release_displays()
```

5) Configure the I2C pins for communication with the OLED display: :

```
1 SCL, SDA = board.GP17, board.GP16
2 i2c = busio.I2C(SCL, SDA)
```

6) Configurar el bus de display y la dirección de la pantalla OLED: :

```
1 display_bus = displayio.I2CDisplay(i2c,
2     device_address=0x3C)
```

7) Configurar el display SSD1306 con dimensiones 128x64

```
1 WIDTH = 128
2 HEIGHT = 64
3 display = SSD1306(display_bus, width=
    WIDTH, height=HEIGHT)
```

B. Funciones

1) Funcion del display:

```
1 def imprimir(text,text_2,x,y):
2
3     splash = displayio.Group()
4     display.root_group = splash
5     color_bitmap = displayio.Bitmap(WIDTH
        , HEIGHT, 1)
6     color_palette = displayio.Palette(1)
7     color_palette[0] = 0x000000 # Negro
8
9     bg_sprite = displayio.TileGrid(
10         color_bitmap, pixel_shader=
11             color_palette, x=0, y=0)
12     splash.append(bg_sprite)
13     text = text
14     text_area = label.Label(terminalio.
15         FONT, text=text, color=0xFFFFF, x=
16             x, y=y)
17     splash.append(text_area)
18     i=0
```

2) Funciones para controlar el carro:

```
1 def detener_carrito():
2     m11.value = False
3     m12.value = False
4     m21.value = False
5     m22.value = False
6
7 def mover_adelante():
8     m11.value = True
9     m12.value = False
10    m21.value = True
11    m22.value = False
12
13 def mover_atras():
14     m11.value = False
15     m12.value = True
16     m21.value = False
17     m22.value = True
```

```
def girar_izquierda():
    m11.value = True
    m12.value = False
    m21.value = False
    m22.value = False

def girar_derecha():
    m11.value = False
    m12.value = False
    m21.value = True
    m22.value = False
```

3) Búfer para capturar imágenes de la cámara:

```
buf = bytearray(2 * cam.width * cam.
    height)
```

C. Ciclo

1) Medidor de diferencia:

```
buf = bytearray(2 * cam.width * cam.
    height)
while True:
    cam.capture(buf)
    matrix = []
    for j in range(cam.height):
        row = []
        for i in range(cam.width):
            if ((buf[2 * (cam.width * j +
                i)]*10//255)<5):
                intensity="0"
            else:
                intensity="-"
            row.append(intensity)
        matrix.append(row)

    %Procesar la ltima lnea de la
    imagen para determinar la
    direccin
    ultima = matrix[-1][:]
    suma = 0
    contador = 0
    for i in range(len(ultima)):
        if ultima[i] == "0":
            suma += i
            contador += 1

    promedio = suma / contador if
        contador != 0 else 0
    diferencia = (20 - promedio) * (100 /
        20) if promedio != 0 else 100
    diferencia = (diferencia / 2) + 50
    pwm.duty_cycle = abs(int((diferencia
        * 65535) / 100))
```

```
30     print(f'Diferencia: {diferencia}')
```

```
1 if diferencia > 60:
2     girar_derecha()
3 elif diferencia < 40:
4     girar_izquierda()
5 else:
6     mover_adelante()
7 i=1+i
```

```
1 speed_1=random.randint(20,120)
2 speed_2=random.randint(2,12)*10-3
3
4 imprimir("Rueda 1:"+str(speed_1),"
5         Rueda 2:"+str(speed_2),25,30)
6
7
8 time.sleep(0.1)
```

D. PROBLEMAS A SOLUCIONAR

Es de destacar ciertos aspectos que se presentaron al poner en funcionamiento el sistema.

- Es el caso de la velocidad de los motores, la cual depende de la señal PWM que entrega el microcontrolador. Esto ocasionó el inconveniente de sincronizar y ajustar los valores adecuados para el seguimiento de la línea ya que deberían funcionar lo suficientemente lento para permitir al procesador analizar la siguiente imagen dada por la cámara.
- También se presentó el inconveniente de conectar los motores y la cámara simultáneamente a un único microcontrolador. Es de aclarar que la cámara está programada en Circuit Python mientras que los motores se habían configurado para micropython. Esto debido al proceso que estuvimos siguiendo a lo largo del curso. La solución a estos inconvenientes fue trabajar todo el código el entorno Circuit Python asumiendo un único script para todo el sistema.
- La potencia que se requería a partir de las baterías utilizadas, las cuales debían ser portátiles y muy ligeras, exigía un nivel de precisión bastante alto ya que podrían verse comprometidos los demás componentes del sistema. Esto se solucionó con un puente H dividiendo la potencia en diferentes secciones con valores específicos para cada uno de los componentes.

CONCLUSIÓN

La integración del aprendizaje por refuerzo (RL) en el automóvil robótico de seguimiento de líneas presentó numerosos desafíos, principalmente debido a la complejidad y las demandas de recursos de los algoritmos de RL. El aprendizaje por refuerzo requiere una amplia recopilación de datos, potencia computacional para el entrenamiento y un

ajuste continuo de los hiperparámetros, lo que resultó estar más allá de las capacidades del microcontrolador Raspberry Pi Pico. La memoria y la potencia de procesamiento limitadas del microcontrolador dificultaron la implementación y ejecución de modelos RL sofisticados que pudieran aprender y adaptarse en tiempo real a varios escenarios de seguimiento de líneas.

A pesar del plan inicial de utilizar RL para el aprendizaje adaptativo y la toma de decisiones, las limitaciones nos llevaron a adoptar un enfoque más tradicional y determinista para la tarea de seguir líneas. Al centrarnos en técnicas sencillas de procesamiento de imágenes y un control motor preciso, pudimos lograr un comportamiento confiable de seguimiento de líneas. La estrategia implicó capturar imágenes de la cámara OV7670, procesar la última línea de la imagen para detectar la posición de la línea y ajustar la dirección y velocidad del automóvil en función de esta información.

Para mejorar la precisión y coherencia de la detección de líneas, se colocaron LED cerca de la cámara. Estos LED proporcionaron condiciones de iluminación constantes, reduciendo las sombras y mejorando la capacidad de la cámara para discernir la línea en la pista. Esta modificación mejoró significativamente la solidez del algoritmo de seguimiento de línea, asegurando que el automóvil pudiera detectar y seguir la línea de manera confiable incluso en condiciones de luz ambiental variables.

Además, era esencial ajustar cuidadosamente las velocidades del motor para lograr un control suave y receptivo. Al ajustar las señales PWM enviadas a los motores, podríamos controlar con precisión la velocidad y el índice de giro del automóvil. Este ajuste aseguró que el automóvil pudiera seguir la línea suavemente sin movimientos ni desviaciones abruptas, lo que resultó en un desempeño más estable y predecible.

En resumen, si bien la integración del aprendizaje por refuerzo no fue factible debido a limitaciones del hardware, la combinación de procesamiento de imágenes, iluminación controlada con LED y ajustes meticulosos de la velocidad del motor permitieron que el automóvil robótico siguiera una línea con éxito. Este enfoque demostró ser eficaz y práctico dadas las limitaciones, lo que demuestra la importancia de aprovechar los recursos disponibles y adaptar las estrategias para lograr los resultados deseados.

REFERENCES

- [1] Adafruit. "Adafruit SSD1306 OLED Display Library". [Online]. Available: <https://github.com/adafruit/Adafruit-CircuitPython-SSD1306>
- [2] Adafruit. "Adafruit OV7670 Camera Library". [Online]. Available: <https://github.com/adafruit/Adafruit-CircuitPython-OV7670>
- [3] Raspberry Pi Foundation. "Raspberry Pi Pico Documentation". [Online]. Available: <https://www.raspberrypi.org/documentation/microcontrollers/>
- [4] Pin Out Raspberry Pi Pico W [online]. Available: <https://core-electronics.com.au/guides/raspberry-pi-pico-w-overview-features-specs/>