Informe: Grupo de Programación de la Raspberry de Sistemas Embebidos

David Alejandro Gutiérrez Martínez - 20201005084

Felipe Junior Gonzalez Mojica - 20201005042

Jenson David Cuastumal Almario - 20181005

Juan David Godoy Barragan – 20192005116

Omar Fabian Forero Uriza – 20201005188

RESUMEN

Se implementó un sistema de control en lazo abierto para motores DC, usando PWM y puentes H. Se añadieron servomotores en el brazo robótico, configurados mediante señales PWM específicas para definir el ángulo de posición. La visualización se realizó con una pantalla OLED SSD1306, y se integró una cámara OV7670 para captura de imágenes. Para manejar secuencias de movimientos, se utilizó el formato JSON, que facilita la lectura y ejecución ordenada de comandos. Toda la estructura fue pensada para permitir control paso a paso, ajustes de velocidad, dirección y posicionamiento, logrando un sistema embebido funcional, claro y fácil de modificar.

Palabras clave : Control en lazo abierto, PWM, motores DC, servomotores, JSON, Raspberry Pi Pico W, sistema embebido, pantalla OLED, cámara OV7670.

ABSTRACT

This project used different parts to control a mobile car and a robotic arm. The motors use PWM to control speed and direction. The arm uses servomotors with specific pulse width to set the angle. An OLED screen shows information and the OV7670 camera can take pictures. To send the instructions, the JSON format is used. This makes easy to send commands in steps, with all the data organized. Everything runs with a Raspberry Pi Pico W and Python. The system works good, and it's easy to test and change.

***Keywords*— Open-loop control, PWM, DC motors, servomotors, JSON, Raspberry Pi Pico W, embedded system, OLED display, OV7670 camera..**

I. INTRODUCCIÓN

En este proyecto se trabajó con distintos componentes de un sistema embebido, enfocado en el control de un carro móvil y un brazo robótico. Para lograrlo, se integraron motores DC, servomotores, una pantalla OLED, una cámara digital y un sistema de comunicación basado en JSON. Todo esto fue controlado desde una Raspberry Pi Pico W usando MicroPython. A lo largo del desarrollo, se combinaron técnicas básicas como el control por PWM, junto con estructuras de datos que permiten enviar instrucciones paso a paso de forma organizada. Esta integración permite que el sistema funcione de forma modular, interpretable y ajustable según las condiciones físicas del entorno y del hardware disponible.

II. MARCO TEÓRICO

A. CONTROL EN LAZO ABIERTO PARA LOS MOTORES DEL CARRO.

Los motores de corriente continua (DC) son ampliamente utilizados en robótica debido a su simplicidad de control y bajo costo. Su velocidad y dirección pueden ser controladas fácilmente mediante señales PWM y circuitos de inversión de polaridad como los puentes H.

- La velocidad del motor se controla variando el ciclo de trabajo (duty cycle) de la señal PWM.
- La dirección se controla invirtiendo la polaridad de la tensión aplicada al motor

La PWM (Pulse Width Modulation) es una técnica usada para controlar la energía entregada a una carga, como un motor. Consiste en una señal digital que varía su ciclo de trabajo (el porcentaje del tiempo que la señal está en nivel alto).

En la Raspberry Pi Pico W, el PWM puede configurarse fácilmente usando MicroPython con el módulo `machine`.

Para controlar un motor con la Pico W y el L298N, se requiere:

- Conectar dos pines GPIO de la Pico a los pines IN1 e IN2 del L298N para cada motor (dirección).
- Conectar un pin GPIO con capacidad PWM al pin ENA o ENB (velocidad).

A. CONTROL DE LOS SERVOMOTORES DEL BRAZO.

El uso de servomotores en aplicaciones de automatización, robótica y control electrónico ha cobrado gran relevancia gracias a su capacidad de posicionamiento preciso. A diferencia de los motores DC convencionales, que solo permiten girar en un sentido o en otro con control de velocidad, el servomotor incorpora un mecanismo de realimentación interna que le permite alcanzar y mantener una posición angular específica. Este mecanismo se basa en un potenciómetro interno acoplado al eje del motor, que junto con un circuito de control, compara continuamente la posición deseada con la real, ajustando el movimiento del eje hasta alcanzar el punto esperado.

Para comunicarle al servomotor cuál es la posición deseada, se utiliza una técnica llamada modulación por ancho de pulso (PWM, por sus siglas en inglés). El PWM no transmite una señal analógica continua, sino una secuencia de pulsos digitales cuyo tiempo de duración —o ciclo de trabajo— determina el valor interpretado por el dispositivo. En el caso de servomotores de rotación estándar, la señal PWM posee una frecuencia fija de 50 Hz, es decir, un periodo de 20 milisegundos. Dentro de cada periodo, el ancho del pulso (usualmente entre 1 ms y 2 ms) define el ángulo que tomará el eje del servomotor. Por convención, un pulso de 1 ms representa una posición de 0 grados, uno de 1.5 ms representa 90 grados (posición intermedia) y uno de 2 ms representa 180 grados. Aunque estos valores pueden variar ligeramente según el fabricante del servo, este intervalo suele mantenerse como referencia general.

B. PANTALLA OLED PARA INFORMACIÓN DEL CARRO.

Una pantalla OLED de **128x64 píxeles** es uno de los modelos más comunes usados en electrónica embebida y proyectos de prototipado. Suele tener una **diagonal de 0.96 pulgadas** y una resolución suficiente para mostrar texto, íconos y gráficos simples.

Características típicas:

- Resolución: 128 columnas × 64 filas
- Color: Monocromática (generalmente blanca, azul o amarilla)
- Interfaz: I2C o SPI (los modelos I2C son muy populares)
- Chip controlador: **SSD1306**

El **SSD1306** es un controlador gráfico de pantalla que administra la memoria de video y genera las señales necesarias para controlar la pantalla OLED.

Funciones clave:

- Comunicaciones I2C o SPI
- Memoria de pantalla integrada (GRAM)
- Soporte para modos de página (paginado)
- Control del brillo, contraste y encendido/apagado

Gracias a este chip, el microcontrolador no necesita manejar directamente los 8192 píxeles (128×64), sino que se comunica con el SSD1306 usando comandos de alto nivel.

C. CÁMARA OV7670

Las cámaras digitales modernas utilizan sensores electrónicos para capturar imágenes. Los más comunes son los sensores CCD (Charge-Coupled Device) y CMOS (Complementary Metal-Oxide-Semiconductor). Estos sensores convierten la luz en señales eléctricas que posteriormente son digitalizadas. El sensor CMOS, como el integrado en la cámara OV7670, es ampliamente utilizado en sistemas embebidos debido a su bajo consumo de energía, alta velocidad de operación y bajo costo.

La OV7670 es un sensor de imagen digital desarrollado por OmniVision. Está basado en tecnología CMOS y diseñado para capturar imágenes a color. Su resolución es de 640×480 píxeles (VGA), con una tasa de hasta 30 cuadros por segundo. La salida puede configurarse en diferentes formatos como RGB565, YUV o YCbCr, y la comunicación con el controlador externo se realiza mediante una interfaz paralela de 8 bits. La configuración interna del módulo se gestiona mediante el protocolo SCCB (Serial Camera Control Bus), una variante de I²C.

En funcionamiento, el sensor captura imágenes mediante una matriz de píxeles sensibles a la luz. La luz incidente se convierte en señales eléctricas analógicas que luego son digitalizadas internamente. La OV7670 incorpora funciones de procesamiento como control automático de ganancia, ajuste de exposición, balance de blancos y reducción de ruido. Estos parámetros pueden ser modificados por el usuario a través de comandos enviados por el bus SCCB.

Para comunicarse con un microcontrolador, la OV7670 utiliza señales de sincronización como VSYNC (inicio de cuadro), HSYNC (inicio de línea) y PCLK (reloj de píxel). La imagen se transmite por un bus de datos paralelo de 8 bits (D[7:0]), que debe ser leído y almacenado por el microcontrolador a una velocidad suficiente para evitar pérdida de datos. Esto implica la necesidad de un manejo eficiente de memoria y temporización, especialmente en plataformas con recursos limitados.

La cámara OV7670 es utilizada en numerosos proyectos de bajo costo relacionados con visión artificial, procesamiento de imágenes embebido, sistemas de vigilancia y robótica.

D. JSON PARA EL CONTROL POR PASOS

JSON (JavaScript Object Notation) es un formato de texto ligero utilizado para representar datos estructurados. Está basado en una sintaxis simple de pares clave-valor y listas ordenadas. Es fácil de leer y escribir para las personas, pero también fácil de interpretar por las máquinas, lo cual lo ha convertido en uno de los formatos más usados para el intercambio de información entre sistemas [1].

A diferencia de otros formatos como XML, JSON no requiere etiquetas de apertura y cierre, ni estructuras complejas. Su estructura es clara y directa: se usan llaves {} para agrupar elementos y corchetes [] para listas. Esto lo hace mucho más compacto y más fácil de procesar. Además, es compatible con una gran variedad de lenguajes de programación, incluyendo Python, JavaScript, C++, entre otros [1].

El uso de JSON es muy común cuando se necesita enviar datos entre aplicaciones que se comunican por red, por ejemplo, entre una interfaz web y un servidor, o entre un dispositivo embebido y una computadora. Gracias a que es texto plano, puede ser enviado fácilmente por HTTP, sockets o incluso almacenado en archivos para configuraciones o registros de actividad [2].

Otra ventaja de JSON es su flexibilidad. Se puede representar información simple como un número o un texto, pero también estructuras más complejas como listas de objetos, arreglos anidados o datos jerárquicos. Esto permite adaptarlo a una gran variedad de aplicaciones, desde el almacenamiento de configuraciones hasta el envío de datos en tiempo real.

En sistemas donde se manejan datos estructurados, JSON también facilita el mantenimiento del código. Al representar la información de forma clara, permite que otros desarrolladores o usuarios entiendan rápidamente cómo están organizados los datos, sin necesidad de documentación extensa.

Aunque JSON no incluye validación de esquema por sí mismo, existen herramientas externas que permiten definir qué estructura debe tener un documento JSON válido. Esto ayuda a evitar errores cuando se esperan ciertos campos o formatos específicos.

En resumen, JSON es un formato de intercambio de datos que combina simplicidad, legibilidad y versatilidad. Por estas razones, se ha vuelto un estándar de facto en el desarrollo de aplicaciones modernas que requieren comunicación estructurada entre diferentes partes de un sistema.

III. DESARROLLO

A. CARACTERIZACIÓN DE LOS SERVOMOTORES PARA EL BRAZO ROBÓTICO:

Para el brazo robótico, es necesario caracterizar experimentalmente cada servomotor. El objetivo es que, dado un ángulo de entrada, se determine el valor adecuado del ciclo útil de la señal PWM. Esta caracterización se realiza aplicando distintos valores de ciclo útil, posicionando el brazo en el ángulo deseado y asignando el ciclo útil correspondiente a dicho ángulo. En la siguiente imagen se muestran los ángulos asignados al brazo robótico. El ángulo del hombro está definido entre el segmento L1 y la base, mientras que el ángulo del codo se define entre los segmentos L1 y L2.

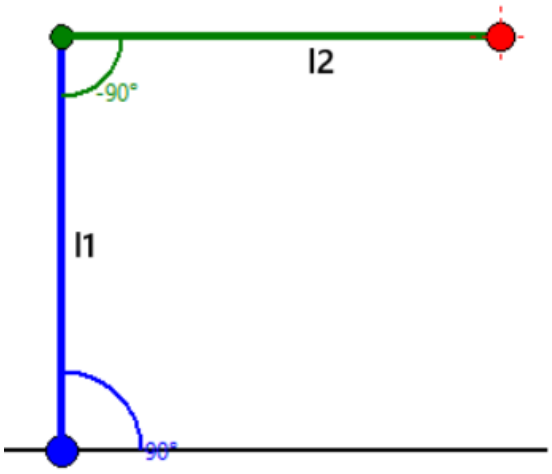

Figura. Ángulos de referencia para el brazo robótico.

Como se observa, el segmento L1 toma como referencia la base, mientras que L2 toma como referencia a L1. Cuando estos segmentos son perpendiculares, el ángulo entre ellos corresponde a 90°.

La base del brazo robótico tiene su ángulo de referencia (0°) alineado con el centro del carro. Desde esta posición, puede girar hacia la derecha hasta aproximadamente +90° y hacia la izquierda hasta –90°, como se muestra en la siguiente imagen (vista superior).

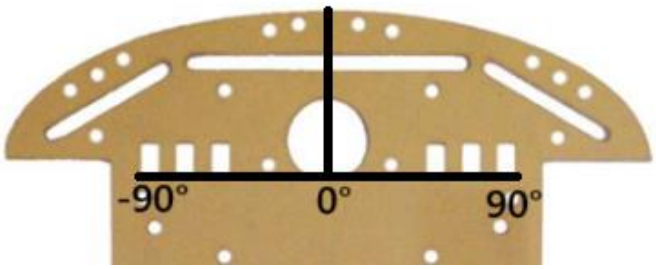

Figura. Ángulos de referencia para la base del brazo robótico.

Una vez obtenidos los ángulos de referencia para el brazo robótico y su movimiento de tres grados de libertad, se procede a determinar las funciones características de cada servomotor. Cada función se representa mediante una ecuación donde, como se mencionó anteriormente, la entrada es un valor angular (entre 0° y 90° para los servomotores del codo y hombro, y entre –90° y 90° para el servomotor de la base), y la salida corresponde al

ciclo útil del PWM necesario para posicionar el servomotor en el ángulo deseado.

Dado que los servomotores presentan un comportamiento aproximadamente lineal, pueden caracterizarse mediante una ecuación lineal de la forma:

$$y = m \cdot x + b$$

En donde "**y**" es el ciclo útil del PWM en nanosegundos, "**x**" el valor del ángulo en grados, "**m**" pendiente de la recta y "**b**" intersección con el eje Y.

Finalmente, debido a las características físicas del brazo robótico, existen casos inusuales donde el ángulo del codo no corresponde exactamente con el valor programado. Un ejemplo notable ocurre cuando el hombro está en 0°: en esta posición, el servomotor del codo adopta una configuración invertida. Para corregir este comportamiento, es necesario implementar una excepción en el código que compense dicha inversión.

Estas particularidades hacen necesario realizar pruebas experimentales adicionales, con el objetivo de identificar todos los casos especiales y aplicar las correcciones correspondientes en el código de control de los servomotores.

B. Cámara OV7670:

El módulo de cámara digital **OV7670** fue integrado en un sistema embebido basado en la placa **Raspberry Pi Pico W**, con el objetivo de capturar imágenes en tiempo real y procesarlas o transmitirlas para fines de monitoreo o visión artificial. La OV7670 es un sensor CMOS de bajo costo que permite capturar imágenes a color en distintas resoluciones y formatos de salida, siendo adecuada para proyectos de procesamiento de imágenes en plataformas con recursos limitados.

La inicialización de la cámara se realiza mediante la función `init_camera()`, que establece la comunicación entre el microcontrolador y el sensor utilizando el protocolo **SCCB** (Serial Camera Control Bus), una variante del I²C. A través del bus I²C, se configuran los registros internos del sensor, definiendo el modo de operación, formato de imagen y parámetros de captura.

En este caso, se utilizaron pines digitales de la Raspberry Pi Pico para conectar el bus de datos paralelo de 8 bits (D0 a D7) y las señales de control necesarias: **PCLK** (reloj de píxel), **VSYNC** (inicio de cuadro), **HREF** (inicio de línea), así como los pines de **RESET** y **PWDN (power down)**. También se genera una señal de reloj externa (**MCLK**) desde el microcontrolador para sincronizar el funcionamiento del sensor.

Una vez establecida la conexión física y lógica, el sensor es configurado con las siguientes instrucciones:

- `cam.wrapper_configure_yuv()` establece el formato de salida como **YUV422**, donde cada grupo de dos píxeles se representa con cuatro bytes. Este formato es útil para la compresión ligera y procesamiento de color eficiente.
- `cam.wrapper_configure_base()` ajusta parámetros básicos del sensor como brillo, ganancia, control automático de exposición y sincronización.
- `cam.wrapper_configure_size(OV7670_WRAPPER_SIZE_DIV2)` define el tamaño de imagen reducido (QVGA: 320x240 píxeles), lo que permite una buena relación entre resolución y velocidad de captura.
- `cam.wrapper_configure_test_pattern(OV7670_WRAPPER_TEST_PATTERN_NONE)` desactiva cualquier patrón de prueba, habilitando la captura de imágenes reales del entorno.

La función `send_image()` realiza la adquisición directa de una imagen completa a través de `cam.capture(buf)`, donde `buf` es un arreglo de bytes del tamaño adecuado para almacenar la imagen en el formato YUV422. La captura se sincroniza con las señales VSYNC y PCLK para asegurar que los datos sean leídos correctamente del bus paralelo.

La elección del formato YUV y del tamaño QVGA se debe a consideraciones de eficiencia: un tamaño mayor implicaría un aumento considerable en el uso de memoria y tiempo de captura, lo cual podría afectar la estabilidad del sistema embebido. Además, la interfaz paralela requiere una lectura rápida y sincronizada de datos, lo cual implica una correcta coordinación entre hardware y software para evitar pérdida de información.

Este diseño permite capturar imágenes periódicamente desde la OV7670 y almacenarlas o transmitirlas para aplicaciones como vigilancia remota, reconocimiento de objetos o seguimiento visual. La estructura modular del código, basada en la clase `OV7670Wrapper`, facilita la reutilización y el ajuste de parámetros según las necesidades de cada aplicación.

C. CALIBRACIÓN DE MOTORES:

La librería se diseñó para los siguientes casos del movimiento:

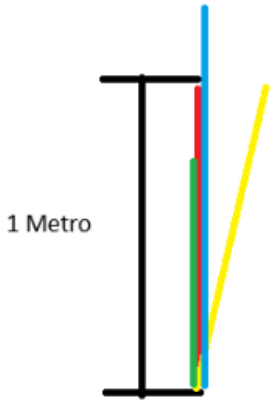

Figura. Posibles trayectorias vs trayectorias Deseadas.

La trayectoria roja es la deseada, la librería permite ajustar el valor de velocidad para los casos de la trayectoria verde y azul, en donde el carro no alcanza el metro o en el caso en que el carro se pase del metro, además, existe un ajuste para cada motor, el cual permite ajustar el ciclo útil para controlar los casos como los de la trayectoria amarilla, en los cuales como un motor genera más empuje que el otro con el mismo PWM genera esta diagonal. Esto se logra variando los valores de las variables: CM_POR_SEGUNDO o VELOCIDAD_BASE.
Las funciones **init, y** _set_motors se encargan de la preparación inicial de los motores, estableciendo dirección y frecuencia del PWM a utilizar.

La función _ajustar_velocidad se emplea para modificar la velocidad del carro en los momentos en que este realiza las maniobras.

mover_adelante, mover_atras, son para avanzar y retroceder.

En el caso de las curvas, se contemplan los siguientes escenarios:

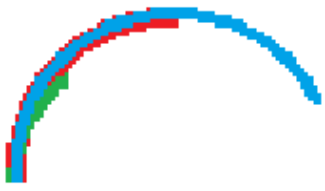

Figura. Posibles trayectorias vs trayectorias Deseadas.

Sea la trayectoria Roja un giro deseado de 90 grados, se contemplan los casos en que el carro realizará un giro mayor, como el que ilustra el azul o el caso en que se realice un giro más corto como el verde, en el código se pueden realizar las calibraciones necesarias para ajustarse a las condiciones físicas de los motores y realizar los giros como el usuario lo desea.

Las funciones girar_izquierda y girar_derecha, reciben como entrada solamente el angulo de giro deseado. Mientras que curva_suave permite seleccionar el radio y el sentido de giro , además de la distancia que se debe recorrer.

D. PANTALLA OLED:

Para hacer uso de esta pantalla, se emplearon librerías básicas de github, sin embargo, la librería My_oled _lib si fue diseñada para el carro. La cual se encarga de:

MyOLED_**init_** Es utilizada para inicializar la comunicación con la pantalla OLED, está destinada para conexiones i2c, si se requiere OSI, es posible, pero se debe realizar las modificaciones correspondientes.

clear se usa para limpiar el display, write_text se usa para escribir texto o cadenas de texto. Las funciones _draw, tienen como variantes pixel, rectangle y line, las cuales dibujan en la pantalla dichas figuras según las coordenadas que desee el usuario.

_Display_on y display_off se encargan de encender o apagar la pantalla momentaneamente.

E. JSON PASO A PASO

Se tiene la función parsear_comando, la cual recibe un texto en formato JSON que contiene una secuencia de instrucciones. El objetivo de esta función es convertir ese texto en una lista de comandos que se puedan ejecutar uno por uno, en orden y con los datos validados.
Lo primero que se hace es usar json.loads() para convertir el texto en un diccionario de Python. Si el JSON no es válido o viene vacío, se lanza un error y se termina la ejecución.
Luego, se busca una clave principal que empiece con "Carro_", por ejemplo "Carro_1". Esa clave es la que contiene todos los pasos del comando. Si no se encuentra, también se genera un error. Después de obtener esa sección, se revisa que tenga contenido.
A continuación, se identifican todas las claves que representan pasos, es decir, aquellas que empiezan por "Paso_". Estas se ordenan numéricamente para asegurar que los pasos se ejecuten en la secuencia correcta, sin depender del orden en el que venían originalmente.
Se recorre cada paso ya ordenado y se verifica que tenga una sección llamada "Movimiento". Dentro de esta sección deben estar los campos distancia_mm, velocidad_mm_s y radio_mm. Si falta alguno de ellos, se muestra un error indicando qué dato está incompleto.

F. Con esos datos se construye un diccionario para el comando actual. Se guarda la distancia original, su valor absoluto, y la velocidad. También se define si el movimiento será hacia adelante o hacia atrás, dependiendo del signo de la distancia.

G. Después se analiza el campo radio_mm. Si el valor es "inf" (infinito), se asume que el movimiento es recto. Si es un número, se interpreta como un giro, se calcula el ángulo y se

define si el giro es hacia la izquierda o la derecha, dependiendo del signo. También se asigna una velocidad de giro, ya sea la que viene en el JSON o un valor por defecto de 60 grados por segundo.
La función también revisa si en el paso hay una sección de "Brazo". Si está presente, se valida que existan los tres ángulos (angulo0_grados, angulo1_grados, angulo2_grados) y que sean numéricos. Si todo está bien, se guardan esos valores junto con el tiempo de ejecución del servo (t_ser), que puede estar definido o asumir un valor por defecto de 1 segundo. Si no hay instrucciones para el brazo, estos campos se dejan en None.
Cada comando generado se agrega a una lista llamada secuencia_comandos. Al final, esa lista se devuelve para ser usada por el sistema de control. En caso de cualquier error en el proceso, se imprime un mensaje y se devuelve None, para evitar que se ejecuten comandos malformados o incompletos.

IV. Resultados

H. Diagrama de clases

Para realizar el algoritmo que controla el carro y el brazo robótico, se plantea el siguiente diagrama de clases, en donde cada clase se encarga

de realizar una única actividad, para después ser llamado por el main.

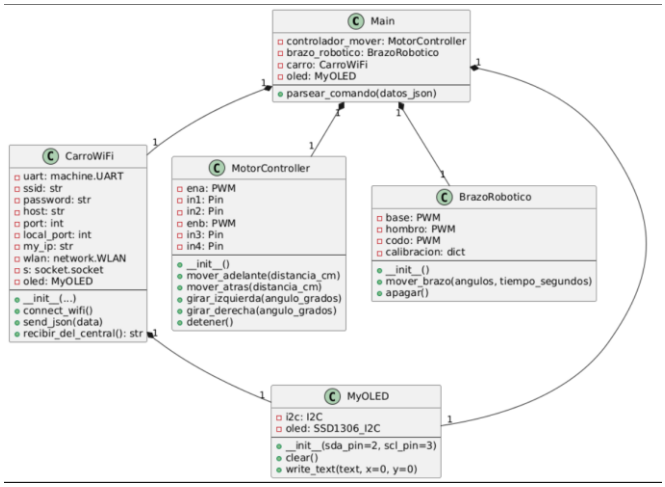

Figura. Diagrama de clases.

I. Funciones de caracterización para el brazo robótico

Para obtener la ecuación característica de cada servomotor, se registran múltiples puntos de ciclo útil junto con sus ángulos correspondientes, como se muestra en la siguiente tabla.

Ángulo (°)	Ciclo útil (ns)
0	1500000
90	550000

Tabla. Datos de ángulos y ciclo útil para el servomotor del **Hombro**.

En la tabla anterior se seleccionaron dos puntos importantes para caracterizar el servomotor que controla el movimiento del **hombro**. Con estos datos, se generó un gráfico en Excel que permitió determinar la ecuación característica del servomotor, como se muestra en la siguiente imagen.

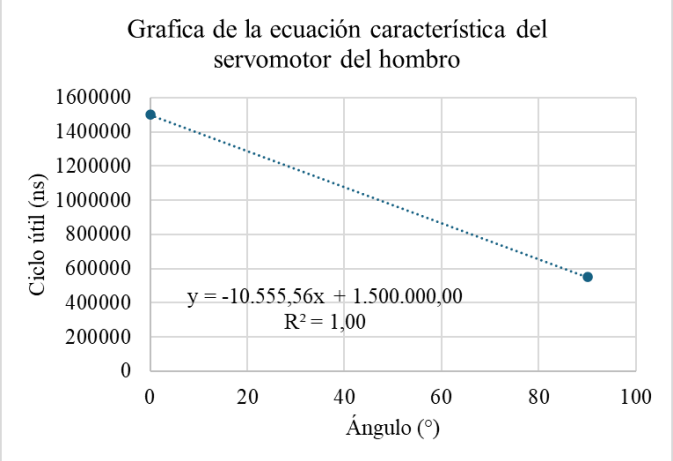

Figura. Gráfica y función característica del servomotor del hombro.

Este mismo procedimiento se repitió para los servomotores del codo y la base. A continuación, se presentan las tablas correspondientes, junto con sus gráficas y ecuaciones características resultantes.

Ángulo (°)	Ciclo útil (ns)
20	1150000
90	2200000

Tabla. Datos de ángulos y ciclo útil para el servomotor del **Codo**.

Para el ángulo mínimo del codo, se establece un margen de seguridad debido a las limitaciones físicas del brazo robótico, que le impiden alcanzar exactamente los 0°.

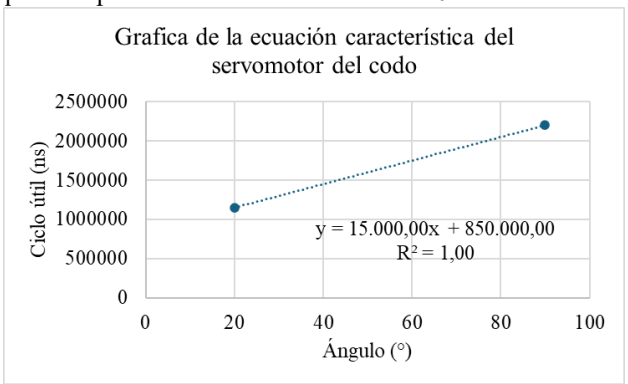

Figura. Gráfica y función característica del servomotor del codo.

Finalmente, para el servomotor de la base se tiene:

Ángulo (°)	Ciclo útil (ns)
-90	620000
0	1500000
90	2500000

Tabla. Datos de ángulos y ciclo útil para el servomotor de la **Base**.

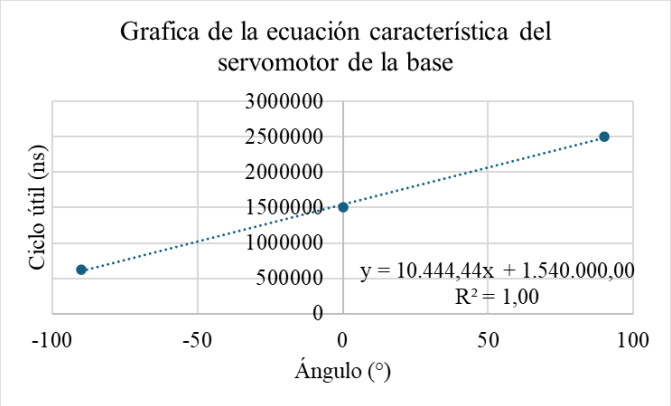

Figura. Gráfica y función característica del servomotor de la base.

Es de gran relevancia destacar que tanto las funciones de caracterización de los servomotores como las excepciones requeridas son únicas para cada brazo robótico. Esto se debe a variaciones en la respuesta de los servomotores ante diferentes ciclos útiles y a diferencias mecánicas durante el ensamblaje: el piñón de cada servo puede fijarse en posiciones angulares distintas al acoplar las piezas del brazo.

J. Ejemplos fotos realizadas con Cámara ov7670:

A continuación se presentan ejemplos de imágenes capturadas utilizando la cámara OV7670 conectada a la Raspberry Pi Pico W. Las fotografías fueron tomadas bajo diferentes condiciones de iluminación y distancia para evaluar el desempeño del sensor en entornos reales.

Las imágenes fueron transmitidas exitosamente a un servidor remoto mediante una conexión WiFi, utilizando el protocolo TCP/IP. La resolución de transmisión fue de 320×240 píxeles (QVGA), optimizando así el tamaño del buffer y la velocidad de envío. En el servidor, se realizó el procesamiento de los datos en formato YUV422, incluyendo su conversión a imágenes en color y la aplicación de un modelo de superresolución (EDSR) basado en redes neuronales, que permitió escalar las imágenes hasta una resolución de 1280×960 píxeles con una notable mejora visual. Finalmente, las imágenes procesadas fueron almacenadas y presentadas a través de una interfaz web.

Estas evidencias visuales permiten verificar el correcto funcionamiento del sistema de adquisición, transmisión y mejora de imágenes, sirviendo como base para futuras optimizaciones en resolución, compresión y análisis en tiempo real.

Figura. Ejemplo 1 Imagen capturada por la OV7670 con superresolución aplicada

Figura. Ejemplo 2 Imagen capturada por la OV7670 con superesolucion aplicada

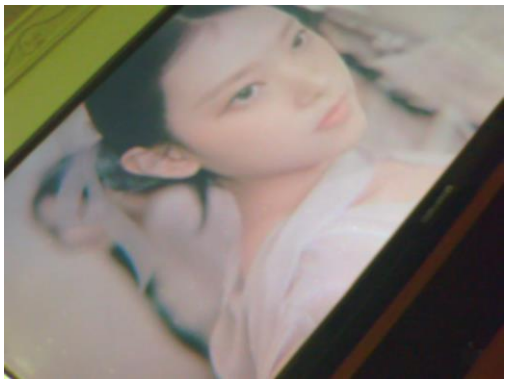

Figura. Ejemplo 3 Imagen capturada por la OV7670 con superesolucion aplicada

K. Diagrama de flujo para la clase del control del brazo robótico

Una vez obtenidas las funciones características de cada servo y detalladas las excepciones que presenta el brazo, se desarrolla la clase que contiene el algoritmo de control del brazo robótico.

Los siguientes diagramas de flujo ilustran la lógica a seguir para su implementación.

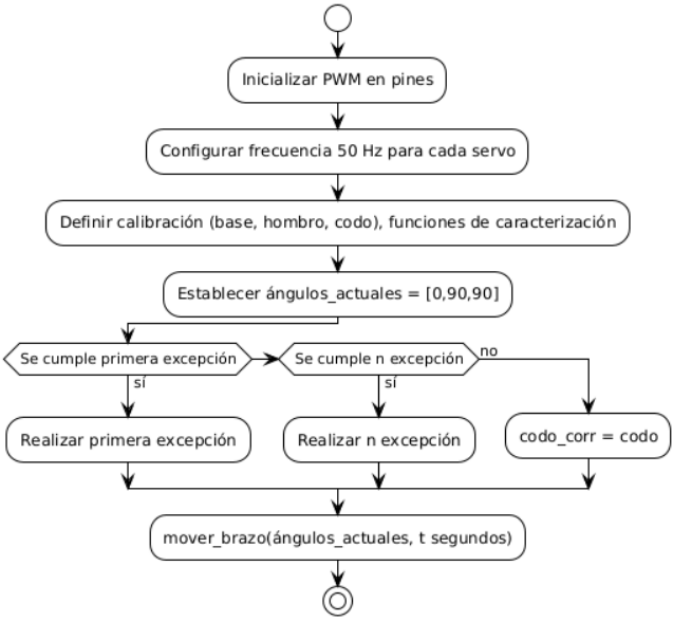

Figura. Lógica principal

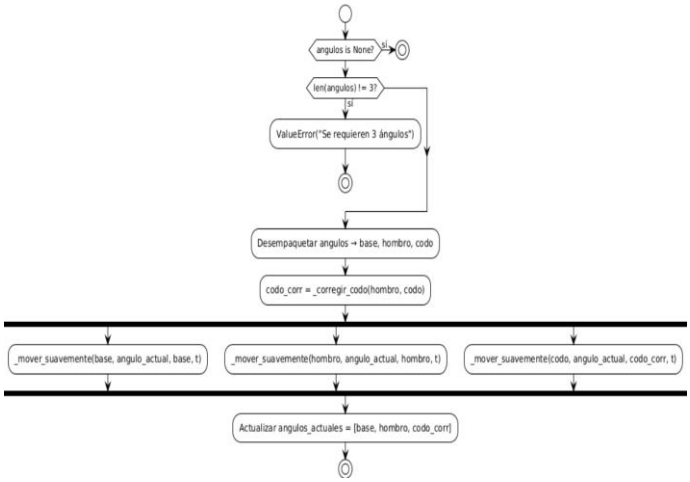

Figura. Función “mover_brazo”

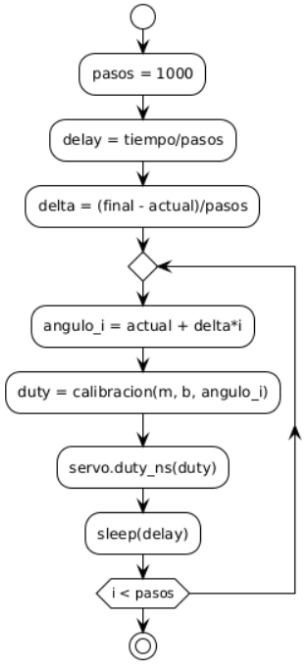

Figura. Función “mover_suavemente”

L. DIAGRAMA DE FLUJO PARA CONTROL DE LOS MOTORES.

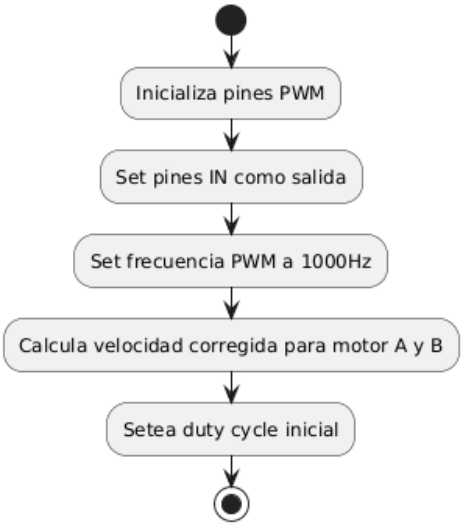

Figura. Función “MotorController.init”

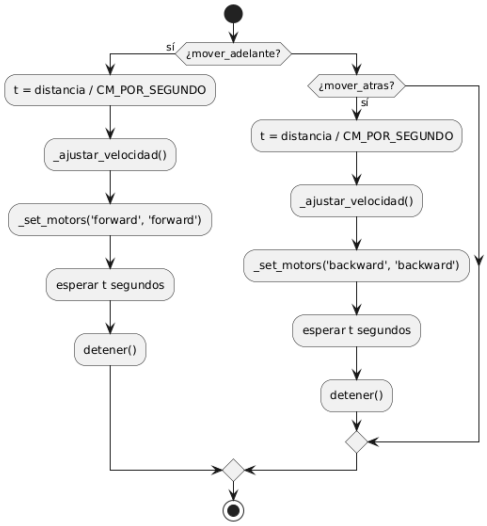

Figura. Funciones MotorController mover adelante, mover atrás

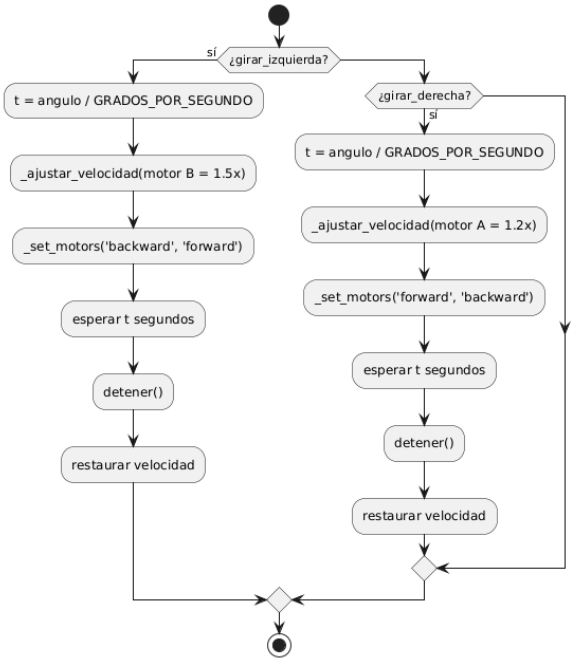

Figura. Funciones MotorController girar derecha, girar izquierda.

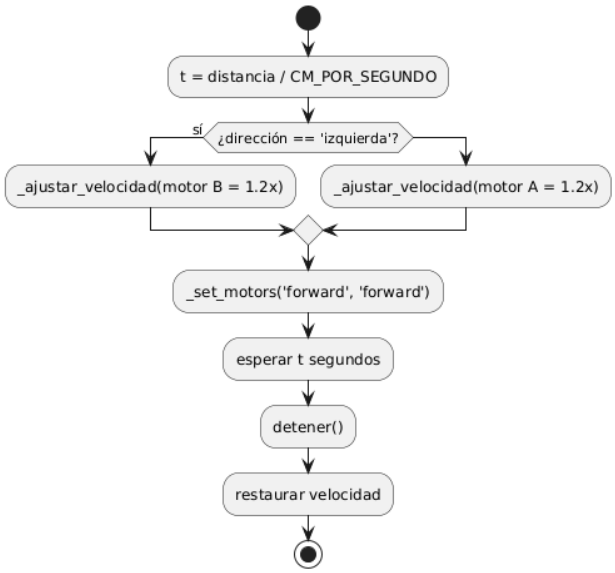

Figura. Función MotorController curva suave.

M. DIAGRAMA DE FLUJO PARA CÁMARA OV7670

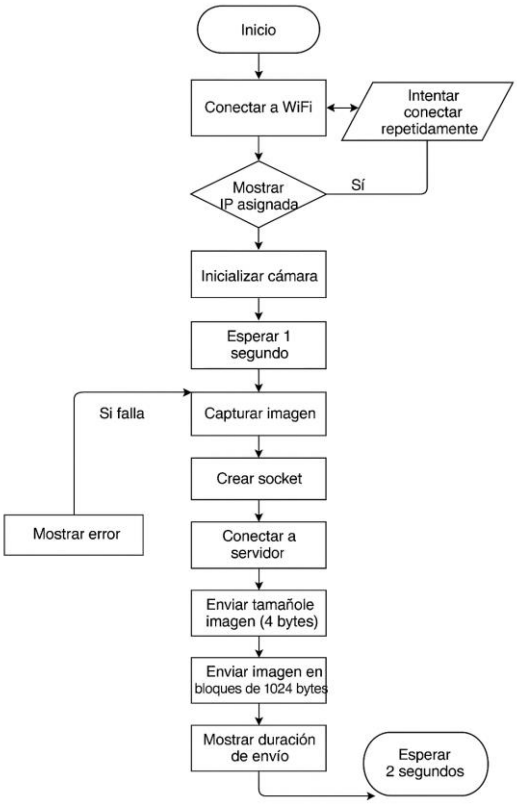

Figura. Diagrama de flujo cámara

Jkfnkdjfd

N. DIAGRAMA DE FLUJO PARA OLED.

Figura. Funcion my_oled_lib inicializacion

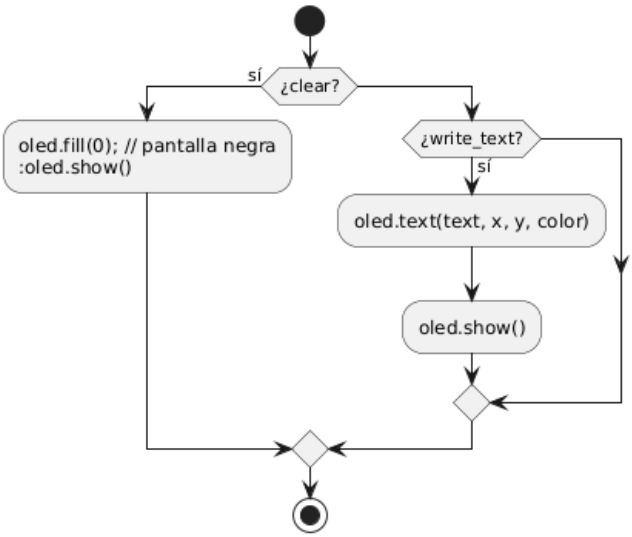

Figura. Funciones my_oled_lib write_text y clear

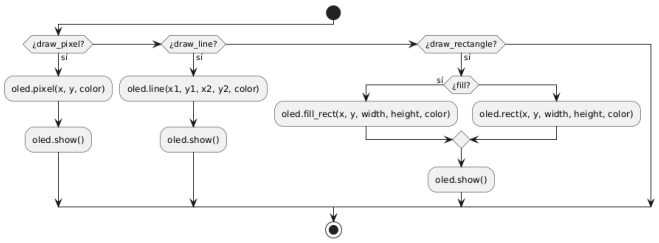

Figura. Funciones my_oled_lib draw line, pixel y rectangle.

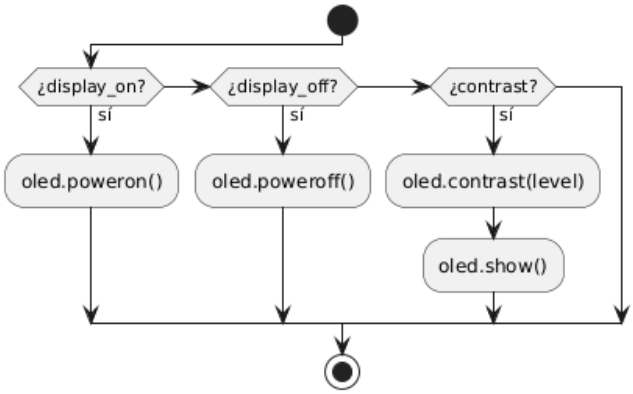

Figura. Funciones display_on, display_off y contras

O. DIAGRAMA DE FLUJO DEL JSON

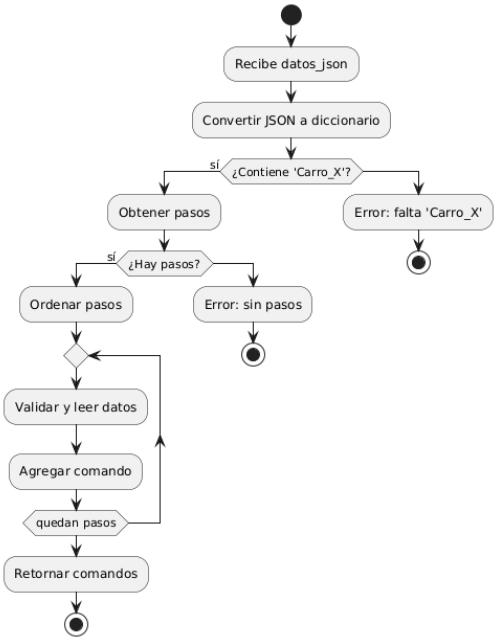

Figura. Funcion del JSON(Parsear_comando)

IV. CONCLUSIONES

- A lo largo del proyecto se logró integrar y controlar varios módulos, como motores DC, servomotores, una pantalla OLED y una cámara OV7670, todos manejados desde una Raspberry Pi Pico W. Aunque el sistema terminó funcionando como se esperaba en la mayoría de los casos, llegar a ese punto no fue fácil. Hubo varios retos durante el proceso, especialmente al momento de hacer que todo trabajara de forma estable y coordinada.

∉ El uso de JSON fue útil para estructurar los comandos paso a paso, permitiendo que el carro y el brazo ejecutaran acciones en orden y con los datos bien definidos. Sin embargo, hacer que el código interpretara correctamente toda la estructura tomó tiempo y pruebas, sobre todo al validar campos y

asegurar que los pasos se siguieran en el orden correcto.

- El control por PWM permitió manejar tanto la velocidad como la dirección de los motores, así como la posición de los servomotores del brazo. En ambos casos, fue necesario hacer ajustes para compensar diferencias físicas y mejorar la precisión. Aun con eso, hubo movimientos que no salieron perfectos, pero el comportamiento general fue aceptable.

- La pantalla OLED se comportó bien y fue sencilla de integrar. La cámara, en cambio, requirió más trabajo. Aunque se lograron capturas de imagen, su manejo dependía mucho del código, los tiempos y el uso eficiente de la memoria.

- En resumen, el sistema funcionó, aunque no de forma perfecta. Se cumplieron los objetivos principales y se identificaron mejoras posibles para una siguiente versión. La experiencia permitió entender mejor cómo combinar hardware y software en un sistema embebido real, con sus ventajas y también con sus limitaciones.

REFERENCIAS

[1] T. Bray, "The JavaScript Object Notation (JSON) Data Interchange Format," RFC 7159, Internet Engineering Task Force, 2014. [Online]. Available: https://tools.ietf.org/html/rfc7159

[2] O. Lassila and D. L. McGuinness, "The role of frame-based representation on the Semantic Web," W3C Submission, 2001. [Online]. Available: https://www.w3.org/Submission/2001/SUBM-daml+oil-reference-20010327/