Universidad de Cantabria

Facultad
de
Ciencias

CONTROL DE UN ROBOT CUADRÚPEDO DESDE M2OS

(Control of a quadruped robot from M2OS)

Trabajo de Fin de Grado
para acceder al
GRADO EN INGENIERÍA INFORMÁTICA

Autor: Héctor Bringas López

Director: Mario Aldea Rivas

Co-Director: Héctor Pérez Tijero

Septiembre – 2025

Resumen

El robot Bittle, fabricado por Petoi, es un robot cuadrúpedo compacto diseñado para la educación en robótica y el desarrollo de proyectos experimentales. Este proyecto tiene como objetivo desarrollar una librería ligera para el control del Bittle, para su ejecución en el sistema operativo en tiempo real M2OS. Se planea portar la funcionalidad de la librería original OpenCat, la cual presenta la desventaja de ocupar prácticamente toda la memoria, manteniendo el control de servomotores y sensores.

Para lograrlo, se analizarán los componentes esenciales de OpenCat y se adaptarán sus funciones a M2OS mediante wrappers en C++ y Ada, de manera que puedan integrarse con programas escritos en Ada y C.

El proyecto permitirá que Bittle ejecute acciones moduladas, respondiendo a comandos predefinidos, y abrirá la posibilidad de futuras mejoras, como la integración de nuevos sensores que harán que reaccione a situaciones imprevistas.

Palabras clave:

Librería, Bittle, Robot, Arduino, M2OS

Abstract

The Bittle robot, manufactured by Petoi, is a compact quadruped robot designed for robotics education and experimental project development. This project aims to develop a lightweight library for controlling Bittle, optimized for execution on the M2OS real-time operating system. The plan is to port the functionality of the original OpenCat library, which has the disadvantage of occupying practically all the available memory on the robot, while maintaining control of servos and sensors.

To achieve this, the essential components of OpenCat will be analysed and its functions adapted to M2OS through wrappers in C++ and Ada, enabling integration with programs written in Ada and C.

The project will allow Bittle to execute modulated actions, responding to predefined commands, and will open the possibility for future enhancements, such as integrating new sensors that will enable it to react to unforeseen situations.

Key Words:

Libraries, Bittle, Robot, Arduino, M2OS.

Índice

1. Introducción

1.1 Motivación

La empresa Petoi ha desarrollado un robot cuadrúpedo denominado Bittle, controlado por una placa basada en el microcontrolador Arduino Uno [1]. Arduino es una plataforma de hardware abierto que incluye microcontroladores programables y un entorno de desarrollo fácil de usar, muy popular para proyectos de electrónica y robótica. Para facilitar la programación del robot, la empresa proporciona la librería OpenCat [2], diseñada para controlar las funciones y movimientos de Bittle. Bittle es un robot cuadrúpedo que utiliza esta tecnología para ofrecer movilidad y ejecución de tareas robóticas, mientras que OpenCat facilita su programación mediante una interfaz y un conjunto de funciones predefinidas. El análisis de dicha librería en un trabajo previo [3] ha puesto de manifiesto sus elevados requisitos de memoria y su deficiente estructura interna.

Por otro lado, dentro del grupo ISTR se ha desarrollado el sistema operativo M2OS. M2OS es un sistema operativo muy sencillo pensado para soportar aplicaciones concurrentes en microcontroladores con recursos de memoria muy limitados (como es el caso del Arduino Uno).

En microcontroladores con recursos limitados, la concurrencia resulta muy útil porque permite la gestión y mantenimiento de tareas, de modo que varias funciones críticas (como controlar motores o procesar sensores) puedan ejecutarse sin bloquearse entre sí. También aporta flexibilidad, al permitir que cada módulo se implemente como una tarea independiente, lo que facilita desarrollo, depuración y ampliación, y posibilita un uso eficiente de los recursos al equilibrar CPU y memoria para optimizar la respuesta en tiempo real.

En este proyecto se propone la implementación de un controlador para el robot Bittle mediante un conjunto de tareas M2OS, con el objetivo de gestionarlo de manera directa, sin utilizar OpenCat. Asimismo, se contempla la integración de dispositivos adicionales que permitan mejorar la interacción del robot tanto con las personas como con su entorno.

1.2 Objetivos

El objetivo de este proyecto es desarrollar una nueva librería que sustituya a OpenCat. Esta nueva librería estará diseñada específicamente para el sistema operativo de tiempo real M2OS.

A partir de este objetivo general, se plantean los siguientes subobjetivos:

- Entender el funcionamiento y extraer los componentes principales de la librería OpenCat de cada funcionalidad.
- Desarrollar controladores para los dispositivos específicos de Bittle (acelerómetro y controlador de servo motores).
- Controlar el movimiento del robot mediante tareas dentro de M2OS usando la librería desarrollada.

2. Herramientas, tecnologías y metodología

2.1 Framework Arduino

Arduino [4] es una plataforma habitual en el ámbito de la electrónica y la programación, destacándose por su enfoque en la accesibilidad y la versatilidad. Desde su creación en 2005 en el Interaction Design Institute Ivrea, Italia, ha evolucionado para convertirse en una herramienta esencial tanto en entornos educativos como profesionales.

A diferencia de otras plataformas, Arduino integra hardware y software de código abierto [5], permitiendo a los usuarios diseñar y construir dispositivos interactivos que responden a estímulos del entorno. Su arquitectura abierta fomenta la colaboración y el intercambio de conocimientos, lo que ha dado lugar a una comunidad global activa y en constante crecimiento.

Las placas Arduino, están equipadas con microcontroladores que gestionan las operaciones del dispositivo. Estas placas ofrecen múltiples pines de entrada y salida, facilitando la conexión con sensores, actuadores y otros componentes electrónicos. En este sentido, Arduino se destaca por su gran adaptabilidad gracias al uso de módulos electrónicos, que permiten ampliar las funcionalidades de las placas sin necesidad de diseñar circuitos complejos desde cero. Estos módulos son pequeños dispositivos listos para conectarse directamente a los pines de la placa, lo que facilita la incorporación de nuevas características a los proyectos. Existen módulos para comunicación como Bluetooth, Wi-Fi, para interacción con el entorno como sensores de todo tipo, pantallas para salida de datos, así como otros módulos de control. Gracias a su diseño y a la disponibilidad de librerías específicas para cada componente, estos módulos pueden integrarse de forma rápida y sencilla. Esto permite a los usuarios crear sistemas personalizados, en muy poco tiempo. Esta modularidad convierte a Arduino en una plataforma ideal para experimentar, aprender y desarrollar soluciones reales en una amplia variedad de ámbitos.

En este proyecto, Arduino resulta esencial porque proporciona la base de programación del robot Bittle, permitiendo un control accesible de servos y sensores, así como la incorporación de módulos adicionales para ampliar sus capacidades.

El entorno de desarrollo integrado de Arduino (IDE) [6] permite escribir, compilar y cargar código en las placas de manera sencilla. El lenguaje de programación utilizado es una adaptación simplificada de C++, lo que facilita el aprendizaje y la implementación de proyectos, incluso para aquellos sin experiencia previa en programación. Los programas en Arduino siguen una estructura básica compuesta por dos funciones principales: setup() y loop(). La función setup() se ejecuta una sola vez al iniciar el programa y se utiliza para

configurar aspectos iniciales, como la configuración de pines de entrada o salida. Por su parte, la función loop() contiene el código que se repite continuamente mientras la placa esté en funcionamiento, permitiendo así el control y monitoreo constante del hardware.

2.2 Placa Arduino Uno

El microcontrolador Arduino Uno (Figura 1) es una placa de desarrollo basada en el microcontrolador ATmega328P es la base utilizada por Petoi para desarrollar su microcontrolador para el robot. Este modelo de placa es ampliamente utilizado debido a su simplicidad, robustez y facilidad de uso, lo que la hace ideal tanto para desarrolladores principiantes y expertos.

Este microcontrolador opera a una frecuencia de 16 MHz y cuenta con una memoria flash de 32 KB, 2 KB de SRAM y 1 KB de memoria EEPROM, características suficientes para una amplia variedad de proyectos de automatización, control y prototipado.

En cuanto a las entradas y salidas, la placa cuenta con 14 pines digitales de entrada/salida, de los cuales 6 pueden ser utilizados como salidas de modulación por ancho de pulso (PWM) importantes para este proyecto, pero insuficientes para un robot de las características del Bittle. Además, tiene 6 pines de entrada analógica.

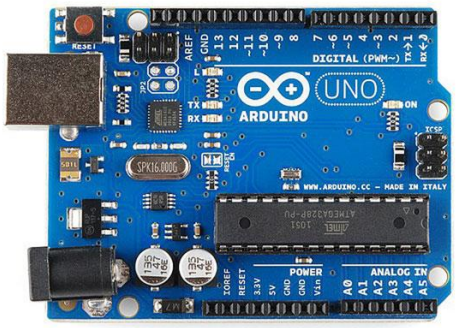


Figura 1- Placa Arduino Uno [7]

2.3 Placa NyBoard V1_2

La NyBoard V1_2 [8] es la versión de la placa Arduino Uno desarrollada por Petoi para sus robots cuadrúpedos, visible en las Figuras 2 y 3 por ambos lados.

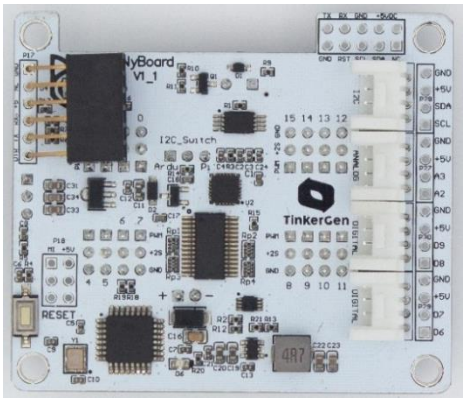


Figura 2 - NyBoard V1_2 Frontal

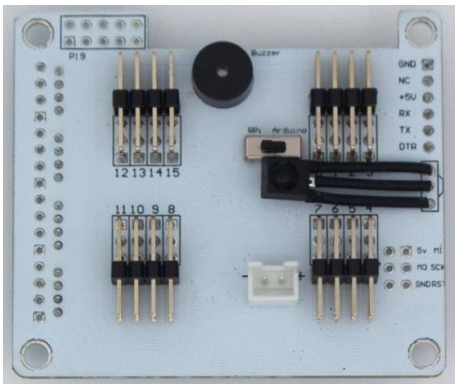


Figura 3 - NyBoard V1_2 Trasera

Aunque mantiene el mismo microcontrolador ATmega328P a 16 MHz, y por tanto es totalmente compatible con Arduino Uno, la NyBoard es fundamental en este proyecto, ya que amplía las prestaciones del Arduino Uno:

- Control de servos mejorado: integra el controlador PCA9685, ofreciendo 16 canales de salida PWM de 12 bits, superando los 6 PWM nativos del Arduino Uno.

- Sensor de movimiento: incluye un MPU6050 con acelerómetro y giroscopio, junto con un procesador digital de movimiento, lo que permite obtener datos precisos sobre la orientación y el estado dinámico del robot.

- Mayor almacenamiento: dispone de una EEPROM externa de 8 KB para guardar parámetros y datos de calibración, frente a los 1 KB de EEPROM interna del Uno.

- Conmutador I2C: mediante el PCA9306, permite alternar entre Arduino y Raspberry Pi como maestro I2C, facilitando la integración híbrida.

- Conectividad ampliada: soporta módulos de comunicación como USB, Bluetooth y Wi-Fi, ofreciendo más opciones que el Arduino Uno.

- Soporte para periféricos: cuenta con cuatro conectores Grove que simplifican la conexión de sensores y módulos adicionales.
- Gestión de energía: incorpora un conector de batería y un sistema de detección de voltaje, proporcionando una alimentación más segura y adecuada para sistemas móviles.
- LED RGB: permite indicar estados y generar efectos visuales sin necesidad de componentes externos, a diferencia del LED integrado del Arduino Uno que no es RGB.
- Zumbador (Buzzer): facilita la generación de sonidos y alertas en proyectos interactivos.
- Receptor infrarrojo: permite recibir señales de mandos a distancia, ampliando las opciones de control remoto.

En las imágenes del Anexo I se pueden observar algunos de los componentes mencionados ubicados en la placa.

2.4 Servomotor

Un servomotor es un actuador electromecánico diseñado para posicionar un eje angular con alta precisión y control. A diferencia de un motor de corriente continua convencional, el servomotor no gira libremente, sino que se mueve hasta alcanzar un ángulo determinado gracias a la integración de un motor eléctrico, un sistema de engranajes reductores, un potenciómetro que mide la posición y una electrónica de control que compara la señal de entrada con la posición real del eje.

El control de un servomotor se realiza mediante una señal PWM (Pulse Width Modulation). En este caso, se considera una frecuencia de 60 Hz, equivalente a un periodo de 16,6 ms por ciclo. La posición angular del eje depende únicamente del ancho del pulso, mientras que la frecuencia permanece constante.

Para el servomotor Petoi P1S, utilizado en este proyecto, se trabaja con un rango de movimiento de 270° y un rango de señal de 500 µs a 2500 µs, de manera que:

- Pulso mínimo (~0,5 ms / 500 µs) → posición inicial (0°).
- Pulso máximo (~2,5 ms / 2500 µs) → posición final (270°).

2.5 Robot Bittle

En la Figura 4 se muestra el perro robot Bittle [9], un robot cuadrúpedo controlado por la placa NyBoard, la cual constituye la base para la ejecución de todas sus acciones y gestos. Su estructura integra servomotores y componentes diseñados para reproducir con realismo los movimientos naturales de un perro.

Figura 4 – Perro Robot Bittle [10]

Como se puede apreciar en esta imagen, el robot consta de 4 tipos partes:

- Torso: Funciona como estructura central y base para la placa NyBoard, la cual se encuentra debajo de la cubierta negra. Esta sección también contiene el compartimiento para la batería, la cual se encaja mediante una ranura ubicada en el abdomen del robot.

- Cabeza: Es capaz de girar ligeramente hacia ambos lados gracias a un servo integrado en su interior. Además, su diseño permite abrir la boca de forma fija, lo que brinda la posibilidad de colocar accesorios decorativos o bien integrar sensores que detecten obstáculos, lo que amplía sus capacidades.

- Mitades superiores de patas: Estas piezas hacen de nexo entre el torso y las mitades inferiores de las extremidades. Incluyen servos, se conectan a articulaciones móviles tanto por la parte superior como hombros e inferior como caderas.

- Mitades inferiores de patas: Corresponden a las secciones finales de las extremidades del robot. Estas piezas incluyen ranuras para insertar los servos, lo que permite su conexión con las mitades superiores, que funcionan como las articulaciones de codos y rodillas.

La anatomía de animales cuadrúpedos de Petoi, identifica los puntos clave que necesitan movimiento para simular una locomoción natural como se observa en la Figura 5. Así, definen 16 articulaciones controlables.

Figura 5 - Índice de articulaciones del robot [11]

Aunque Bittle toma como referencia la anatomía de un cuadrúpedo real, su estructura está diseñada de forma simplificada. De las 16 articulaciones posibles, el robot utiliza únicamente 7: dos en cada pata (una en la parte superior y otra en la inferior) y una adicional en el cuello. Los servos no siguen una numeración secuencial del 0 al 8, sino que se asignan de acuerdo con su posición dentro del esquema general de articulaciones del sistema, como se observar el siguiente esquema de la Figura 6.

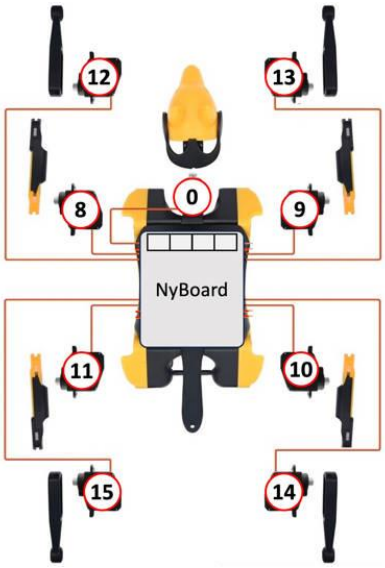


Figura 6 - Cableado de servos [12]

Así, mediante esta estructura y la utilización de los 9 servos, el robot puede imitar los movimientos naturales del animal, llegando a parecerse a un perro auténtico.

El archivo InstinctBittle.h, integrado en el firmware OpenCat y específico para el modelo Bittle, contiene la definición de todas las habilidades (*gaits*) y comportamientos (*behaviors*) del robot. Estas acciones se codifican como arrays de datos en memoria flash, donde cada array representa una animación o patrón de skill.

2.6 Lenguaje C

El lenguaje de programación C se utiliza en este proyecto para programar directamente sobre M2OS. El lenguaje de programación C es uno de los lenguajes de programación más antiguo y ha sido fundamental en la evolución de la informática moderna [13]. Surgido en la década de 1970, fue concebido inicialmente para implementar sistemas operativos. Su diseño lo convierte en una herramienta versátil y eficiente, capaz de operar cerca del hardware gracias a su acceso directo a la memoria, sin dejar de ofrecer estructuras propias de lenguajes de alto nivel.

Esta combinación entre bajo y alto nivel le ha permitido ser ampliamente adoptado en entornos donde el rendimiento y el control del sistema son esenciales, como los sistemas operativos, el software embebido y las aplicaciones críticas [14]. Además, su sintaxis clara y su estructura lógica han influido notablemente en el diseño de lenguajes posteriores, como C++, Java.

2.7 Lenguaje ADA

Ada es un lenguaje de programación inspirado en Pascal, diseñado específicamente para sistemas donde la seguridad y la robustez son esenciales. Su desarrollo estuvo orientado a aplicaciones críticas, como las aeroespaciales o militares [15].

Entre sus principales características se encuentran la fuerte tipificación para una detección temprana de errores y el soporte nativo para programación concurrente. Estas cualidades han consolidado su uso en sectores como la aviación, el control de tráfico aéreo y defensa, donde los fallos no son tolerables y la consistencia del software debe estar garantizada [16]. Basado en estas propiedades, M2OS aprovecha Ada para ofrecer un marco alternativo en microcontroladores pequeños, combinando multitarea y seguridad en el desarrollo de sistemas embebidos

2.8 Sistema operativo de tiempo real M2OS

M2OS [17] es un proyecto de código abierto desarrollado en el grupo ISTR de la Universidad de Cantabria. Originalmente diseñado para Arduino Uno, el sistema cuenta con adaptaciones para procesadores many-core.

M2OS es un sistema operativo en tiempo real de tamaño muy reducido, diseñado específicamente para proporcionar soporte multitarea en microcontroladores con recursos de memoria muy limitados, como el Arduino Uno. Su arquitectura ligera y eficiente permite ejecutar aplicaciones concurrentes en entornos con restricciones de memoria y procesamiento.

Una de sus características principales es el modelo de tareas one-shot no interrumpibles, que permite que todas las tareas compartan un único espacio de pila. Este enfoque optimiza el uso de memoria y simplifica la gestión del sistema, diferenciándolo de otros sistemas operativos en tiempo real más complejos.

Gracias a esta organización en capas, M2OS resulta flexible y escalable, lo que ha permitido su portación desde placas muy simples, como Arduino Uno, hasta arquitecturas de mayor complejidad, como procesadores many-core. Esta separación clara de responsabilidades favorece la portabilidad y demuestra la capacidad del sistema para adaptarse a diferentes contextos embebidos.

M2OS se integra dentro del entorno GNAT Programming Studio (GPS), lo que facilita un flujo completo de desarrollo para placas Arduino. El programador escribe el código en Ada o en C, definiendo las tareas según el modelo de M2OS. Posteriormente, el compilador GNAT, basado en GCC, genera un ejecutable que incluye tanto el núcleo de M2OS como la aplicación del usuario. Este ejecutable se carga en la placa Arduino mediante herramientas de programación tradicionales. El uso de GPS permite además realizar depuración y, en muchos casos, simulación del comportamiento de las aplicaciones, lo que agiliza la prueba y desarrollo de software en tiempo real sobre hardware con recursos limitados.

3. Desarrollo

3.1 Estructura Interna de M2OS

En la organización de directorios de M2OS, dentro de arch/ cada subcarpeta corresponde a una arquitectura específica (por ejemplo, arduino_uno/). En cada una de estas subcarpetas se encuentran al menos las siguientes carpetas:

- hal/: Código de bajo nivel para inicializar y gestionar periféricos, como timers, interrupciones y registros.
- drivers/: Contiene la lógica de los controladores implementados en Ada.
    - libcore/: Incluye los bindings en Ada a los controladores de dispositivo que han sido adaptados a M2OS.
        - wrappers/: Contiene los wrappers de C++ a C.

Al ejecutar el Makefile del directorio libcore/ se genera la biblioteca estática lib_core_arduino.a, que contiene todos los controladores de dispositivos adaptados. Esta biblioteca contiene todo el código necesario para que las aplicaciones puedan ejecutarse sobre la placa Arduino.

3.2 Análisis y funcionamiento de la librería OpenCat

La librería OpenCat es un framework de código abierto diseñado para controlar robots cuadrúpedos. Su función principal es gestionar los movimientos y comportamientos del robot mediante el control de servomotores y el procesamiento de datos de sensores, facilitando una locomoción adaptable.

El núcleo de la librería se encuentra en un conjunto de archivos que incluyen el código principal, normalmente en OpenCat.ino, donde se configuran y ejecutan las rutinas básicas del robot. OpenCat está diseñada para plataformas como Arduino y ESP32.

La librería implementa algoritmos para calcular y controlar las posiciones de las articulaciones de las patas, lo que permite que el robot realice movimientos complejos como caminar, girar y mantener el equilibrio. OpenCat también emplea sensores IMU para medir la orientación y balance del robot en tiempo real. Esto permite estabilizar los movimientos y adaptar la marcha según el terreno o la inclinación, evitando caídas y mejorando la fluidez del desplazamiento.

Además, la librería OpenCat incluye una serie de movimientos preestablecidos que facilitan el control básico y avanzado del robot cuadrúpedo sin necesidad de programar desde cero. Estos movimientos están codificados como secuencias de datos almacenadas en memoria de solo lectura (flash), y se definen en el archivo InstinctBittle.h, obtenido directamente de OpenCat, los movimientos definidos están concretados en la Figura 7. Estas secuencias se denominan skills (gaits o behaviors), y son básicamente arrays de tipo int8_t que contienen la información necesaria para reproducir una postura o animación.

Tipo	Abreviación	Acción
Gait	bdF	Avanzar hacia adelante
	bk	Retroceder
	bkL	Retroceder izquierda
	crF	Girar en círculo adelante
	crL	Girar en círculo izquierda
	jpF	Saltar adelante
	phF	Levantar patas delanteras
	phL	Levantar pata delantera izquierda
	trF	Trotar adelante
	trL	Trotar izquierda
	vtF	Movimiento vertical adelante
	vtL	Movimiento vertical izquierda
	wkF	Caminar adelante
	wkL	Caminar izquierda
Posture	balance	Balancear para equilibrar
	buttUp	Levantar trasero
	calib	Calibración
	dropped	Caído
	lifted	Levantado
	lnd	Aterrizar
	rest	Descansar
	sit	Sentarse
	str	Estirarse
	up	Subir
	zero	Posición cero (neutra)

Behaivour	ang	Movimiento angular
	bf	Movimiento base frontal
	bx	Movimiento base transversal
	chr	Movimiento de giro rápido (carga)
	ck	Patear
	cmh	Movimiento cabeza alta
	dg	Cavar
	ff	Caminar hacia adelante usando sincronización de piernas especial
	fiv	Movimiento libre cinco tiempos
	gdb	Perder el equilibrio
	hds	Sacudir la cabeza
	hg	Mover la cabeza
	hi	Levantar cabeza
	hsk	Saludo con cabeza
	hu	Levantar cabeza hacia arriba
	jump	Saltar
	kc	Movimiento de patada
	mw	Caminata modular
	nd	Inclinarse
	pd	Inclinarse y desplomarse
	pee	Orinar
	pu	Levantar cuerpo
	pu1	Levantar cuerpo variacion
	rc	Movimiento de recuperación
	rl	Rodar
	scrh	Rascarse
	snf	Oler
	tbl	Tabla de equilibrio
	ts	Girar sobre sí mismo
	wh	Sacudir
	zz	Sin acción (Zero Zero)

Figura 7 - Movimientos definidos

Se distinguen dos tipos principales de movimientos según el primer valor del array (byte 0):

- Gait: si el primer valor del array es positivo, define un movimiento.
    - Si el valor es 1, corresponde a una posture, es decir, un movimiento estático, que se ejecuta una sola vez cuando se invoca.
    - Si el valor es mayor que 1, corresponde a un gait propiamente dicho, es decir, una secuencia de varios frames que se repite de manera continua.
- Behaviors: si el primer valor del array es negativo, un movimiento puntual, que se ejecuta solo una vez cuando se invoca.

Un frame es un conjunto de valores que definen la posición de todos los servomotores en un instante específico. Cada uno de esos valores corresponde a un servo distinto, de manera que cuando se aplican todos juntos, el robot adopta una postura concreta.

Por sí solo, un frame representa únicamente una pose fija, similar a una "fotografía" del robot en una posición determinada. Sin embargo, al encadenar varios frames de manera secuencial se obtiene una skill. Estos movimientos pueden clasificarse en gaits o behaviors. En ambos casos, la estructura está compuesta por cuatro bytes iniciales de información de control, seguidos por los frames requeridos para el skill.

Cada skill está estructurado de la siguiente forma [18]:

Byte	Contenido	Explicación
0	Número de frames	Indica cuántas poses (fotogramas) componen la habilidad. Cada pose contiene un conjunto completo de posiciones para todos los servos.
1	Inclinación lateral (roll)	Valor opcional que indica cuánto debe inclinarse el cuerpo lateralmente. Se puede usar para ajustar el equilibrio.
2	Inclinación frontal (pitch)	Similar al roll, pero para inclinación hacia adelante o hacia atrás.
3	Escala de ángulo	Es un factor de multiplicación. Cada valor de servo en los frames se multiplica por este número para convertirlo en grados antes de mover el servo.
4 en adelante	Datos de ángulos por frame	Contiene los valores de posición para cada servo, organizados de manera cíclica. Cada frame incluye un valor por cada servo (del 0 al 15).

Estos datos se almacenan en memoria flash mediante PROGMEM, por lo que deben leerse con la función pgm_read_byte_near().

```
// Behavior: Rascarse (Scratch)
const int8_t scrh[] PROGMEM = {
-6, 0, -30, 1,
 2, 3, 6,
   27,   0, -45,   0, -5, -5,  20, 20,  37, 45, 116,  81, 83,  26, -37,
-26,  8, 0, 0, 0,
   42,   0, -45,   0, -5, -5,  20, 20,  37, 45, 116, -20,  83, 26, -37, -34,
32, 0, 0, 0,
   75,   0, -45,   0, -5, -5,  20, 20,  37, 45, 116, -8,  82, 26, -37, -48,
32, 0, 0, 0,
   72,   0, -45,   0, -5, -5,  20, 20,  41, 45, 116, -25,  72, 26, -37, -25,
32, 0, 0, 0,
   19,   0, -45,   0, -5, -5,  20, 20,  51, 45, 100,  71, 45,  26, -31,
-22,  8, 0, 0, 0,
    0,   0, -45,   0, -5, -5,  20, 20,  45, 45, 105, 105,  45, 45, -45,
-45,  8, 0, 0, 0,};
```

El arreglo scrh define un behavior llamado Rascarse (Scratch) para un robot con múltiples servomotores. Esto se puede entender de la siguiente manera:

- Byte 0 (-6): Número de frames del movimiento (6) y negativo lo que indica que es un behavior.
- Byte 1 (0): Inclinación lateral (roll) del cuerpo. Este valor indica cuánto debe inclinarse el robot hacia los lados, pero se puede ignorar si no se desea ajustar la postura lateral.
- Byte 2 (-30): Inclinación frontal (pitch) del cuerpo. Este valor indica cuánto debe inclinarse el robot hacia adelante o atrás, pero también se puede ignorar si se quiere aplicar los frames tal cual, sin modificar la inclinación general.
- Byte 3 (1): Escala de ángulo; los valores de los servos se multiplican por este número para obtener los ángulos finales en grados.
- Byte 4 en adelante: Cada frame contiene 16 valores, uno por cada servo (del 0 al 15), más algunos ceros al final que podrían representar servos no utilizados o padding.

3.3 Manejadores de dispositivos

Para iniciar el desarrollo del robot, se empleó el entorno Arduino IDE, aprovechando su simplicidad para la carga de programas y la gestión de librerías. A través del gestor de librerías integrado, se localizaron e instalaron las dependencias utilizadas por la plataforma OpenCat, asegurándose de utilizar las

últimas versiones disponibles para evitar problemas de compatibilidad y beneficiarse de las mejoras más recientes.

Entre las librerías incorporadas destacan la PWM Servo Driver de Adafruit, fundamental para el control de los múltiples servomotores mediante el controlador PCA9685, y la librería del sensor MPU6050 de Electronic Cats, que proporciona datos de aceleración y velocidad angular, para el equilibrio y la detección de orientación del robot.

Estas experiencias iniciales resultaron útiles para comprender la dinámica básica del robot y establecer una base sólida para desarrollos posteriores más avanzados. Además de servir ya como desarrollos funciónales para el posterior desarrollo de los módulos para M2OS que posteriormente solo será necesario adaptarlos al entorno del GNAT GPS.

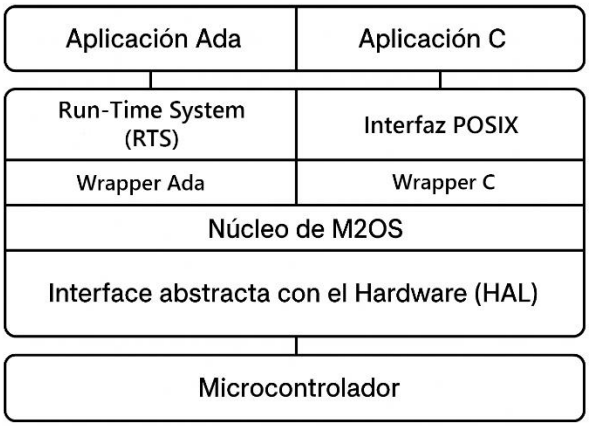
3.3.1 Adaptación de C++ para M2OS mediante wrappers

Un wrapper, o envoltorio, es un componente de software cuya función principal es actuar como una capa intermedia que encapsula otro código, función o librería. Su propósito es simplificar su uso o adaptarlo a distintos entornos sin necesidad de modificar el código original. Gracias a este enfoque, es posible reutilizar software preexistente, adaptar librerías a diferentes lenguajes de programación y garantizar la compatibilidad entre sistemas.

Desde un punto de vista funcional, un wrapper recibe las llamadas de una aplicación, realiza la adaptación necesaria y delega la ejecución en el código subyacente. Esto permite, además, añadir validaciones, gestionar errores o registrar eventos sin alterar la lógica del programa de base.

En el contexto de la integración entre Arduino y M2OS, los wrappers resultan esenciales para lograr la interoperabilidad entre el código escrito en C++, muy común en librerías de Arduino, y el entorno de desarrollo de M2OS, cuyo núcleo está escrito en Ada y que únicamente reconoce interfaces en C o Ada.

Debido a estas diferencias, es necesario crear capas intermedias que traduzcan el código orientado a objetos de C++ a funciones planas compatibles con C, de manera que puedan ser utilizadas en M2OS sin conflictos.



Estos wrappers actúan como adaptadores que:

• Encapsulan funcionalidades escritas en C++ y las exponen como funciones en C que M2OS puede reconocer y ejecutar.

• Gestionan la conversión de tipos de datos y estructuras propias de C++ hacia equivalentes en C.

• Permiten llamar a métodos de clases y objetos desde un entorno procedural, traduciendo dichas invocaciones a funciones C.

• Facilitan la integración de librerías ya existentes en Arduino dentro de proyectos con M2OS, evitando la necesidad de reescribirlas completamente en Ada o C.

Gracias a esta técnica de wrappers, es posible reutilizar librerías y código existentes de Arduino en proyectos con M2OS, minimizando el esfuerzo de reescritura y maximizando la compatibilidad entre ambos entornos.

El propósito de un wrapper es simplificar y estandarizar el acceso a librerías escritas en C++ dentro de entornos que no ofrecen soporte directo para clases u objetos de este lenguaje, como es el caso de M2OS. De esta forma, en lugar de trabajar con interfaces orientadas a objetos, el usuario dispone de funciones planas, fáciles de enlazar y compatibles con C o Ada. Esto permite reutilizar código y librerías ya existentes en C++ dentro de proyectos basados en M2OS, reduciendo el esfuerzo de reescritura y aumentando la compatibilidad entre ambos entornos.

3.3.2 Manejador del acelerómetro MPU6050

El sensor MPU6050 incorpora un acelerómetro y giroscopio en un solo chip y se comunica mediante protocolo I2C. En este proyecto se utiliza para detectar la

inclinación del robot con respecto al eje vertical, lo cual es esencial para mantener el equilibrio o reaccionar ante una caída.

En la función inclinacion(), habiendo inicializado el módulo previamente con su fincion de init, se siguen los siguientes pasos para obtener los valores:

1. Se leen los valores de aceleración cruda en los ejes X, Y y Z con la función:

   imu.getAcceleration(&ax, &ay, &az);

2. Se convierten estas lecturas a unidades de gravedad (g), dividiendo por 16384.0 (valor típico del sensor para 1g).

3. Se calcula el ángulo de inclinación lateral (roll) y hacia adelante/atrás (pitch) utilizando la función trigonométrica de arco tangente:

$$roll = \tan^{-1}(ayf, azf) \cdot \frac{180}{\pi}$$

$$pitch = \tan^{-1}\left(-axf, \sqrt{ayf^2 + azf^2}\right) \cdot \frac{180}{\pi}$$

En resumen, el MPU6050 en el robot Bittle permite percibir su orientación y movimiento, facilitando una ejecución precisa y estable de sus habilidades programadas.

```
#include <MPU6050.h>

extern "C" {
    MPU6050 imu;

    void mpu6050_init() {
        imu.initialize();
    }

    void mpu6050_get_Acceleration(int16_t& x, int16_t& y, int16_t& z) {
        imu.getAcceleration(&x, &y, &z);
    }
}
```

En este wrapper se declara una instancia global de la clase MPU6050. Mediante la directiva extern "C" se exponen funciones planas, accesibles desde M2OS.

- La función mpu6050_init() inicializa el módulo para empezar a realizar lecturas.

- La función mpu6050_get_Acceleration() obtiene las lecturas del

acelerómetro en los ejes X, Y y Z, almacenándolas en las variables pasadas por referencia.

Posteriormente, sobre estas lecturas será necesario realizar los cálculos de conversión a unidades de gravedad (g) y el procesamiento trigonométrico para determinar los ángulos de inclinación lateral (roll) y hacia adelante/atrás (pitch).

3.3.3 Manejador del controlador de sensomotores PCA9685

El PCA9685 es un driver de servomotores de 16 canales con salida PWM (modulación por ancho de pulso) controlable digitalmente a través del bus I2C. Cada canal puede generar una señal PWM independiente, lo que permite controlar 16 servos con un solo microcontrolador.

A continuación, se muestra el wrapper desarrollado para controlar el chip PCA9685, un controlador PWM con 16 canales que suele emplearse en el manejo simultáneo de servomotores. Para integrarla en M2OS se implementa un wrapper en C++, que mediante la directiva extern "C" expone un conjunto reducido de funciones siguiendo la convención de nombres de C. Esto permite que M2OS pueda enlazarlas y utilizarlas sin problemas de compatibilidad.

```
#include <Adafruit_PWMServoDriver.h>

extern "C" {
    Adafruit_PWMServoDriver pca9685 = Adafruit_PWMServoDriver();

    void pca9685_servos_init() {
        pca9685.begin();
        pca9685.setPWMFreq(60);
    }

    void pca9685_set_servo(uint8_t num, uint16_t val) {
        pca9685.setPWM(num, 0, val);
    }
}
```

En este wrapper se declara una instancia global de la clase Adafruit_PWMServoDriver, que constituye la interfaz con el chip PCA9685. A partir de ella, se definen funciones planas que encapsulan los métodos de la clase. La función pca9685_servos_init() inicializa la comunicación con el dispositivo y fija la frecuencia de la señal PWM en 60 Hz, adecuada para el control de servomotores. Por su parte, pca9685_set_servo(uint8_tnum,uint16_t val) ajusta el ciclo útil de la señal PWM en un canal específico, controlando así la posición del servo.

3.4 Funciones de movimiento

El robot cuadrúpedo está diseñado para adoptar distintas posturas y realizar movimientos coordinados. Se diseñaron dos funciones que permiten realizar los movimientos definidos en el fichero InstinctBittle.h: playSkill() y playSkillFrame().

Antes de ejecutar cualquier movimiento, es necesario inicializar el módulo PCA9685 mediante su función init y configurar la frecuencia de PWM a 60 Hz, que es la recomendad, con setFreq(60), que es la frecuencia estándar para servomotores.

La función playSkill() ejecuta un movimiento completo y, para ello, llama internamente a playSkillFrame() para reproducir cada frame del movimiento de forma secuencial. Por su parte, playSkillFrame() permite ejecutar un único frame de manera independiente, ofreciendo un control más detallado.

Gracias a estas funciones, es posible llamar directamente a playSkillFrame() o crear otras funciones que utilicen solo parte de un movimiento. No es necesario ejecutar todo el movimiento de una sola vez: los frames pueden reproducirse de manera ordenada, se puede detener un movimiento en cualquier momento y retomarlo después, o iniciar uno nuevo sin depender de la finalización del anterior. Esto aporta gran flexibilidad en el control del robot.

Función playSkillFrame()

Esta función recibe un puntero a un movimiento y un índice de frame. Su propósito es reproducir un único frame del movimiento, lo que permite ejecutar el movimiento hasta o desde un punto específico, pausarlo y retomarlo posteriormente sin necesidad de reiniciarlo desde el inicio. Para ello realiza los siguientes pasos:

1. En la línea 2 se recupera el valor de escala de ángulo (byte 3).
2. En la línea 8 se accede al bloque de datos correspondiente al frame solicitado.
3. Cada valor leído (de tipo int8_t) representa un ángulo relativo por servo, que se multiplica por la escala para obtener el ángulo real (línea 10).
4. En la línea 11se convierte este ángulo a unidades de señal PWM.
5. Se envía al controlador PCA9685 en la línea 13.

En la variable stopMovement se en carga de parar el movimiento en cualquier momento.

```
void playSkillFrame(const int8_t *skill, uint8_t frameIndex) {
  uint8_t angleRatio = pgm_read_byte_near(skill + 3);
  const int offset = 4 + frameIndex * NUM_SERVOS;

  for (uint8_t s = 0; s < NUM_SERVOS; s++) {
    if (stopMovement) return;

    int8_t raw = pgm_read_byte_near(skill + offset + s);

    int angle = raw * angleRatio;
    int pulse = map(angle, -90, 90, SERVO_MIN, SERVO_MAX);

    pwm.setPWM(s, 0, pulse);
  }
}
```

Los ticks representan las unidades discretas de la señal PWM que controla la posición de un servo a través del PCA9685. Cada tick corresponde a un incremento mínimo en la anchura del pulso, dentro de la resolución del controlador, que es de 12 bits (0–4095). Dado que los servos no interpretan directamente los ángulos en grados, es necesario mapear los valores de ángulo a ticks para generar la señal adecuada.

Para ello, el valor leído del frame, tras multiplicarse por el factor de escala, se obtiene en grados reales. Este ángulo se convierte a ticks mediante un mapeo lineal, relacionando los ángulos con los límites de ticks del servo, utilizando la función map:

ticks = map(angulo_real, angulo_min, angulo_max, SERVO_MIN, SERVO_MAX);

que realiza el cálculo matemático de:

$$\text{ticks} = \text{SERVO\_MIN} + \frac{(angulo\_real - angulo\_min) \cdot (\text{SERVO\_MAX} - \text{SERVO\_MIN})}{angulo\_max - angulo\_min}$$

En esta fórmula:

- SERVO_MIN y SERVO_MAX corresponden a los valores de tick asociados a los límites físicos del servo.
- angulo_real es el ángulo calculado a partir del valor relativo del frame y la escala.
- angulo_min y angulo_max representan el rango máximo de movimiento seguro del servo en grados, que en Bittle va de -135° a 135°.

La determinación de los rangos máximos y mínimos se realiza considerando las características físicas de cada servo. El ángulo mínimo y máximo define los límites mecánicos seguros, que en Bittle van de -135° a 135°, mientras que los

ticks mínimo y máximo se calculan a partir de la duración mínima y máxima del pulso de control del servo (500 µs a 2500 µs) [19], la frecuencia de actualización del PWM (60 Hz) y la resolución del PCA9685 (12 bits, 4096 ticks por período). Cada tick representa un incremento mínimo de pulso de aproximadamente 4,07 µs, por lo que los ticks correspondientes al pulso mínimo y máximo se obtienen como:

$$SERVO\_MIN \ \frac{500\mu s}{4{,}07\mu s} \approx 123$$

$$SERVO\_MAX = \frac{2500\mu s}{4{,}07\mu s} \approx 614$$

Respetar estos rangos es crucial por varias razones: garantiza la seguridad mecánica, evitando daños al servo; asegura la precisión y reproducibilidad del movimiento, manteniendo la correspondencia exacta entre el ángulo deseado y la posición física del servo; y permite una fluidez adecuada a 60 Hz, de manera que los movimientos se reproduzcan de forma continua y suave, sin saltos ni vibraciones.

Función playSkill()

La función playSkill() se encarga de ejecutar una animación completa o postura. Lee los valores de los tres primeros bytes en las líneas 2 a 4, que representan el número de frames y las rotaciones roll y pitch.

Luego identifica si la animación es un behavior (repetición automática) mediante el bloque if en las líneas 8 a 11. Si no es un behavior, al terminar todos los frames, la animación se repite continuamente hasta que se pare3 el movimiento con stopMovement, mientras que, si es un behavior, se ejecuta una sola vez.

A través de iteraciones, se llama internamente a playSkillFrame() para reproducir cada frame de manera secuencial.

Finalmente, la variable stopMovement se reinicia para permitir futuras animaciones.

```
1  void playSkill(const int8_t *skill) {
2    int8_t frameCount = pgm_read_byte_near(skill);
3    int8_t roll = pgm_read_byte_near(skill + 1);
4    int8_t pitch = pgm_read_byte_near(skill + 2);
5
6    bool is_behaivour = false;
7
8    if (frameCount < 0) {
9      frameCount = -frameCount;
10     is_behaivour = true;
11   }
12
13   while (!stopMovement) {
14     for (uint8_t f = 0; f < frameCount; f++) {
15       playSkillFrame(skill, f);
16       delay(50);
17     }
18
19     if (is_behaivour) {
20       break;
21     }
22   }
23
24   stopMovement = false;
25 }
```

3.5 Uso de los dispositivos desde el lenguaje Ada

Una vez elegidas las librerías de Arduino que se desean utilizar para controlar el robot, es necesario adaptarlas si se quieren usar programando en ADA. Para ello, se debe llevar a cabo un proceso conocido como creación de binding. Consiste en encapsular las funciones de las librerías externas dentro de módulos, de forma que se puedan utilizar dentro del sistema.

M2OS está desarrollado en el lenguaje Ada, por lo que no es posible utilizar directamente las librerías escritas en C++. Para resolver esta limitación, se recurre a la creación de interfaces de enlace mediante las cuales Ada puede comunicarse con código C o C++ previamente compilado de las librerías de Arduino.

En este contexto, es necesario crear dos archivos:

Archivo .ads (Ada Specification): Define la interfaz pública del módulo, incluyendo los procedimientos y funciones que estarán disponibles para el resto del sistema. Este archivo cumple una función similar a los encabezados (.h) en C/C++.

Archivo .adb (Ada Body): Contiene la implementación de los elementos definidos en la especificación. Es aquí donde se realizan las llamadas al código externo a través de las directivas pragma Import.

Para facilitar la implementación de los bindings, se utilizaron como referencia otros ya existentes del sistema. Estos archivos proporcionaron ejemplos de estructura, estilo de codificación y métodos adecuados para enlazar código externo. A partir de ellos, se adaptaron las siguientes las librerías anteriormente mencionadas para el control básico del robot.

La librería Adafruit PWM Servo Driver permite controlar múltiples servomotores de manera simultánea. Se encapsularon tres funciones clave para implementar un control básico:

- Inicialización del controlador (init).
- Configuración de la frecuencia de PWM (set_pwm_freq) ya que la utilizada por defecto no es válida para el Bittle.
- Posicionamiento de cada servo mediante valores PWM (set_pwm).

El sensor MPU6050 se utilizaron las siguientes funciones:

- Inicialización del sensor (begin).
- Lectura de aceleración (getAcceleration) para calcular el pitch y el roll.

Para ello lo que se hace es incorporar las funcionalidad la librería "lib_core_arduino.a" de M2OS, que proporciona las funcionalidades básicas necesarias para que el sistema operativo M2OS funcione correctamente sobre placas Arduino.

La integración entre Ada y C/C++ requiere una capa de interfaz que permita invocar funciones escritas en C/C++ desde el código Ada del usuario. Para lograrlo, Ada ofrece el pragma Import, una característica del lenguaje diseñada para facilitar la interoperabilidad con otros lenguajes, en especial con C y C++. Gracias a esta capacidad, es posible utilizar funciones externas como si fueran nativas de Ada, lo que simplifica y agiliza la integración de módulos desarrollados en distintos lenguajes. Por lo que para cada componente se crean los archivos mencionados disponibles en Anexo II.

En los .ads se pueden destacar lar siguientes partes:

- El encabezado package … is establece el espacio de nombres del módulo. La directiva pragma Elaborate_Body garantiza que el cuerpo relacionado (.adb) será elaborado después de la especificación.
- Con procedure se definen únicamente sus cabeceras, sin implementación. La presencia de with Inline_Always sugiere al compilador expandir estas llamadas en el propio lugar de invocación, maximizando

eficiencia. Los parámetros se declaran usando tipos de Interfaces para mantener compatibilidad binaria con C++.

- El size del objeto se puede obtener a partir del sizeof en Arduino, y luego reutilizarlo en Ada para garantizar la correspondencia exacta de memoria.

- Se definen constantes en Ada y se importan valores externos provenientes de librerías C/C++. La cláusula with Import, Convention => C, External_Name => … indica explícitamente que el valor real se obtiene de un símbolo definido en un objeto compilado.

- En la sección privada se declaran los tipos concretos que corresponden a las clases o structs del código C++ original. Se emplea un registro de relleno (array de bytes) para reservar el espacio correcto y luego se aplica el pragma Import (CPP, …) que asocia ese tipo Ada con la clase externa.

- También se define una función que actúa como constructor. Mediante pragma CPP_Constructor se enlaza con el constructor real en C++, identificado por su nombre mangleado. Esto permite instanciar desde Ada objetos de clases escritas originalmente en C++.

- Por último, se declaran instancias globales de los tipos importados (por ejemplo, un controlador de dispositivo) utilizando la palabra clave aliased, lo que hace posible pasar direcciones de esos objetos a funciones que requieran un puntero al propio objeto.

```
with Interfaces.C;

package Simple_Device is
   pragma Elaborate_Body;


   procedure Get_ValueX (X : out Interfaces.Unsigned_16)
     with Inline_Always;

private
   Size_In_Bytes : constant := 8;

   type Filling_Array is array(1 .. Size_In_Bytes) of
Interfaces.Unsigned_8;

   type Device_Type is limited record
      Filling : Filling_Array;
   end record;

   pragma Import (CPP, Device_Type);
   for Device_Type'Size use Size_In_Bytes * 8;
   for Device_Type'Alignment use Standard'Maximum_Alignment;

   function New_Device return Device_Type;
   pragma CPP_Constructor (New_Device, "_ZN..."); -- nombre mangleado del
objeto

   Device_Controller : aliased Device_Type := New_Device;

end Simple_Device;
```

En los .adb se pueden destacar lar siguientes partes:

- procedure: Indica el procedimiento que se quiere obtener en el código Ada. Siempre recibe por parámetro un puntero a sí mismo (This : access ...), que representa al objeto sobre el cual se ejecuta la operación. Junto con ello, se declaran también los parámetros que requiere la función: estos se escriben siguiendo la convención de Ada, pero corresponden directamente a los tipos usados en C++. Para garantizar la compatibilidad entre ambos lenguajes se emplean los tipos definidos en el paquete Interfaces (Unsigned_8, Unsigned_16, etc.), que se mapean a los enteros de tamaño fijo de C++ (uint8_t, uint16_t, etc.). De esta manera, cada parámetro en Ada conserva tanto el tamaño como la semántica del parámetro original en C++.

- with Import: La palabra with, se utiliza para proporcionar información sobre cómo se debe tratar este procedimiento. Al usar Import se especifica que el cuerpo de este procedimiento no se encuentra definido en Ada, sino que está siendo importado desde una fuente externa, en este caso una librería escrita en C++.

- Convention => CPP: Especifica que el lenguaje en el que está escrita la función importada es C++. Esto es necesario porque las convenciones determinan cómo se pasan los parámetros, cómo se gestionan las llamadas.

- External_Name => "_Z…": Esta es la parte fundamental del Import. Aquí se especifica el nombre exacto de la función escrita en C++. En C++, los nombres de las funciones incluyen caracteres adicionales debido a una convención conocida como “name mangling”, que codifica información sobre la firma de la función y otros detalles internos del compilador. Este nombre se obtiene directamente de los archivos .o compilados desde Arduino con las librerías originales, asegurando que el enlace entre Ada y C++ se realice correctamente.

```
-- Declaración de un procedimiento importado desde C++
procedure Procedimiento_Importado(This : access Tipo_Objeto;
                                  Param1 : Interfaces.Unsigned_8;
                                  Param2 : Interfaces.Unsigned_16)
  with Import, Convention => CPP,
       External_Name => "_ZN…";  -- nombre mangleado obtenido del .o de C++
```

Así es como se declaran los procedimientos importados desde C++. A continuación, se implementan unos procedimientos envoltorio (figura 10), que son versiones más fáciles de usar desde Ada. Estos procedimientos llaman

internamente a los procedimientos importados utilizando. Esto permite que desde el código Ada de más alto nivel se invoquen estas funciones de manera directa, simplificando su uso y manteniendo la compatibilidad.

```
-- Procedimiento Ada que envuelve al importado
procedure Procedimiento_Envoltorio is
begin
   Procedimiento_Importado(Objeto_Global'Access, Param1, Param2);
end Procedimiento_Envoltorio;
```

3.6 Desarrollo en M2OS con GNAT GPS

Para familiarizarme con el entorno de desarrollo, inicié con una implementación básica que me permitió comprender la estructura general del proyecto. En cuanto a la programación de aplicaciones, existen diferencias notables respecto al modelo tradicional de Arduino. Mientras que en un programa estándar de Arduino el flujo de ejecución se organiza alrededor de la función setup() (ejecutada una vez) y el bucle loop() (que se repite indefinidamente), en M2OS el paradigma es distinto. El programa principal define un única tarea inicial, la función main. En esta tarea principal, por lo general, no se implementa directamente toda la lógica de la aplicación, sino que se crean y configuran los demás tareas que representarán las distintas tareas concurrentes del sistema. Cada hilo es una tarea independiente con su propio flujo de ejecución, que se coordina gracias al planificador de M2OS

Una vez completados los puntos iniciales, avancé a la fase de compilación y a la integración de los wrappers creados. Junto a ellos adapte el código realizado previamente en Arduino IDE sustituyendo las llamadas a las funciones de los objetos de los módulos por las creadas en los wrappers se encargan de las misma función.

Durante esta etapa fue necesario realizar ajustes en los parámetros de control del servomotor, particularmente en la frecuencia de operación y en los valores derivados de la misma. Tras estas modificaciones, realicé pruebas de movimientos básicos con el fin de evaluar el comportamiento del sistema. Dichas pruebas, que se describen en detalle en secciones posteriores.

Una vez realizado un movimiento simple del motor probe a ver si realizar uno de los movimientos usados por el robot ya que son los que se van a querer utilizar en el control por tareas del Bittle.

4. Pruebas

Dada la delicadeza y el elevado coste del robot, las pruebas iniciales se realizaron sobre una placa original equipada únicamente con un servomotor. Esta decisión introdujo ciertas limitaciones, ya que no se disponía de un entorno completamente realista, lo que dificultaba en parte la evaluación directa del comportamiento del sistema en su conjunto. No obstante, esta metodología permitió avanzar de forma segura y progresiva hacia la validación final.

El proceso de pruebas se estructuró en las siguientes fases:

1. Pruebas con la librería original: se comenzó utilizando la librería proporcionada de manera oficial para la placa, con el fin de evaluar el comportamiento del servomotor en condiciones base. Permitiendo establecer una referencia inicial del funcionamiento esperado.

2. Pruebas con librerías actualizadas y código propio: en una segunda etapa se incorporaron librerías actualizadas junto con código desarrollado específicamente para este proyecto. Se realizaron dos tipos de test:

    - Un movimiento completo del rango del servomotor para comprobar que se era capaz de manejarlo.

```
void test() {
  for (int i = 0; i < NUM_SERVOS; i++) {
    Serial.print("Testing servo: ");
    Serial.println(i);
    pwm.setPWM(i, 0, SERVO_MIN);
    delay(500);
    pwm.setPWM(i, 0, SERVO_MAX);
    delay(500);
    pwm.setPWM(i, 0, (SERVO_MIN + SERVO_MAX) / 2);
  }
}
```

    - La ejecución de uno de los movimientos predefinidos del robot Bittle con las funciones de control creadas. Esta fase permitió comparar los resultados con los obtenidos en la primera prueba, verificando tanto la compatibilidad como la fidelidad respecto al comportamiento original.

    - Para el MPU6050, con la función de obtener inclinación mediante el loop, se observó el movimiento de la placa junto a los resultados obtenidos por la consola. Durante las pruebas, los datos reflejaron de manera coherente los movimientos físicos de la placa: al inclinarla, los valores de inclinación cambiaban según el eje correspondiente, y al mantenerla en posición horizontal, los valores se estabilizaban cerca

de cero. Esto confirma la correcta lectura de inclinación por parte del sensor y la fiabilidad de los datos para aplicaciones de medición de orientación.

```
void inclinacion() {
  int16_t ax, ay, az;

  imu.getAcceleration(&ax, &ay, &az);

  if (ax == 0 && ay == 0 && az == 0) {
    Serial.println("¡IMU sin datos válidos!");
    return;
  }

  const float accelScale = 16384.0;  // 1g = 16384 en MPU6050
  float axf = ax / accelScale;
  float ayf = ay / accelScale;
  float azf = az / accelScale;

  // Calcular inclinación arcotg
  float roll  = atan2(ayf, azf) * 180.0 / PI;
  float pitch = atan2(-axf, sqrt(ayf * ayf + azf * azf)) * 180.0 / PI;

  // Mostrar resultados
  Srial.print("Inclinación =>  Roll: ");
  Serial.print(roll, 2);
  Serial.print("° | Pitch: ");
  Serial.print(pitch, 2);
  Serial.println("°");
}
```

3. Pruebas en M2OS con wrappers desarrollados: posteriormente, se llevaron a cabo pruebas sobre el sistema M2OS, evaluando de nuevo la respuesta del servomotor. Comprobando que los wrappers implementados a partir de las librerías de Arduino funcionaban correctamente, confirmando su viabilidad para controlar el robot bajo este sistema operativo.

    - Un movimiento completo del rango de los servomotor mediante una tarea periódica consiste en desplazarlo de su posición mínima a la máxima (y viceversa).

```
void test() {
  puts("Servo pos:");
  putint(servo_pos);
  puts("\n");

  for (uint8_t s = 0; s < SERVO_NUM; s++) {
    pca9685_set_servo(s, servo_pos);
  }

  if (servo_pos > (SERVO_MIN + SERVO_MAX)/2) {
    servo_pos = SERVO_MIN;
  } else {
    servo_pos = SERVO_MAX;
  }

  TS_INC(next_activation_time, period);
  clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME,
          &next_activation_time, NULL);
}
```

- Asi mismo para el MPU6050 realicé la misma prueba que durante la fase anterior, con la peculiaridad de que las funciones matemáticas en M2OS no están definidas por lo que las tuve que crear yo y los resultados no son igual de exactos.

```
void test_mpu() {
    puts("MPU\n");

    int16_t ax, ay, az;
    mpu6050_get_Acceleration(&ax, &ay, &az);

    const float accel_scale = 16384.0f;
    float axf = ax / accel_scale;
    float ayf = ay / accel_scale;
    float azf = az / accel_scale;

    float roll  = calc_roll(ayf, azf);
    float pitch = calc_pitch(axf, ayf, azf);

    puts("Roll: ");
    imprime_decimal(roll);
    puts(" Pitch: ");
    imprime_decimal(pitch);
    puts("\n");
}
```

5. Conclusiones y futuros desarrollos

5.1 Conclusiones

Este proyecto ha conseguido controlar el robot sin necesidad del controlador del fabricante, lo que representa un avance significativo en términos de flexibilidad y autonomía en su desarrollo. Gracias a la implementación de un conjunto de tareas en el sistema operativo de tiempo real M2OS, se ha demostrado que es posible gestionar de manera directa el funcionamiento del robot Bittle en un entorno distinto al originalmente previsto, ampliando así sus posibilidades de uso e investigación.

Se han integrado dispositivos como el controlador PCA9685 y el sensor MPU6050 en M2OS, y se han desarrollado los wrappers necesarios para utilizarlos desde C como a su vez los bindings para utilizarlos desde Ada. Entre las librerías incorporadas destacan la PWM Servo Driver de Adafruit, para el control de los servomotores mediante el PCA9685, y la librería del sensor MPU6050, utilizada para la adquisición de datos de orientación y aceleración.

El trabajo realizado no solo ha permitido comprender en profundidad la arquitectura del robot, sino también superar los retos técnicos asociados a la integración de código externo en el sistema operativo. Esta experiencia ha resultado especialmente enriquecedora al combinar conceptos de sistemas empotrados, programación de bajo nivel y sistemas operativos de tiempo real en un caso práctico.

Asimismo, se ha sentado un precedente importante para el futuro, ya que la base desarrollada puede ser utilizada como punto de partida para ampliar el sistema, incorporar nuevos sensores o diseñar interfaces que mejoren la interacción del robot con las personas y el entorno. De esta forma, se abre la posibilidad de evolucionar hacia un sistema más personalizable, capaz de responder a necesidades más específicas.

En definitiva, el proyecto no solo ha supuesto un desafío académico y técnico, sino también una oportunidad para contribuir a la evolución del ecosistema de M2OS, reforzando mis conocimientos y permitiéndome avanzar en mi formación en el ámbito de los microcontroladores y sistemas de tiempo real.

5.2 Futuros desarrollos

En etapas posteriores, el proyecto podrá ampliarse mediante las siguientes líneas de trabajo:

1. Pruebas con el robot completo: Una vez ensamblado, se realizarán pruebas de locomoción que permitan comprobar el comportamiento real del sistema y verificar si los desarrollos implementados son correctos y cumplen los objetivos planteados.

2. Desarrollo de más componentes: Se prevé añadir nuevos módulos y sensores, como ultrasónicos o de infrarrojos, que amplíen sus capacidades.

3. Ver diferencias con la librería original: Dado que uno de los objetivos principales fue crear una librería más óptima en cuanto al uso de memoria respecto a la oficial, resulta necesario realizar una comparación detallada.

4. Añadir en M2OS la declaración de los datos en PROGMEM junto con la función de lectura correspondiente, con el objetivo de almacenar todos los movimientos en la memoria flash. Esto permite aprovechar su mayor capacidad en comparación con la SRAM, y así poder guardar todos los movimientos sin problemas de espacio.

Bibliografía

[1] Arduino. Arduino Uno [en línea]. Disponible en: https://www.arduino.cc/en/main/arduinoBoardUno

[2] Petoi, OpenCat-Quadruped-Robot. Disponible en: https://github.com/PetoiCamp/OpenCat-Quadruped-Robot

[3] Odriozola Díaz, Pablo, 2023. Adaptación de la librería OpenCat a M2OS para el control de un robot cuadrúpedo. [tesis]. Universidad de Cantabria. Director: Mario Aldea Rivas, Héctor Pérez Tijero. Disponible en: https://hdl.handle.net/10902/30801

[4] Wikipedia. Arduino. [en línea]. Disponible en: https://es.wikipedia.org/wiki/Arduino.

[5] Xataka. Qué es Arduino, cómo funciona y qué puedes hacer con uno. [en línea]. Disponible en: https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno

[6] Arduino.cl. Programa Arduino [en línea]. Disponible en: https://arduino.cl/programa-arduino/

[7] Electrónica Embajadores, Arduino UNO en: https://www.electronicaembajadores.com/es/Productos/Detalle/LCA1001/modulos-electronicos/arduino/arduino-uno-rev-03-original-a000066/

[8] Petoi. NyBoard v1.1 and v1.2 [en línea]. Disponible en: https://docs.petoi.com/nyboard/nyboard-v1_1-and-nyboard-v1_2

[9] Petoi. Bittle [en línea]. Disponible en: https://bittle.petoi.com/

[10] Petoi, Petoi Bittle Robot Dog. [En línea]. Disponible en: https://www.petoi.com/products/petoi-bittle-robot-dog

[11] Petoi, Petoi robot joint index [En línea]. Disponible en: https://docs.petoi.com/petoi-robot-joint-index

[12] Petoi, NyBoard – Connect the wires. [En línea]. Disponible en: https://bittle.petoi.com/4-connect-the-wires/nyboard

[13] Universitat de València. Lenguaje C. Disponible en: https://informatica.uv.es/estguia/ATD/apuntes/laboratorio/Lenguaje-C.pdf

[14] Dinahosting. Historia del lenguaje de programación C: ¿por qué sigue estando tan presente? [blog], 2024 (aprox.). Disponible en: https://dinahosting.com/blog/historia-del-lenguaje-de-programacion-c-por-que-sigue-estando-tan-presente/

[15] Universidad Nacional del Centro de la Provincia de Buenos Aires. Lenguaje ADA. En: Clases Prácticas – Trabajos Prácticos Especiales [en línea]. Disponible en: https://lengprg1.alumnos.exa.unicen.edu.ar/clases-pr%C3%A1cticas/trabajos-practicos-especiales/ada

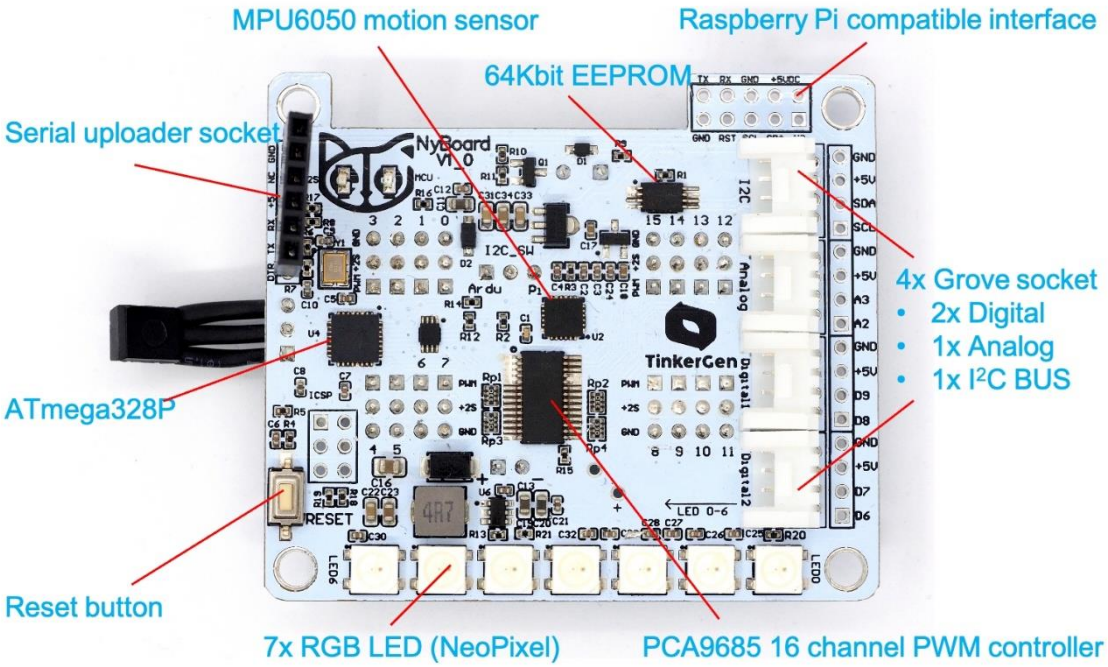
[16] MSMK University. Lenguaje de programación Ada, 4 meses atrás [en línea]. Disponible en: https://msmk.university/lenguaje-de-programacion-ada/

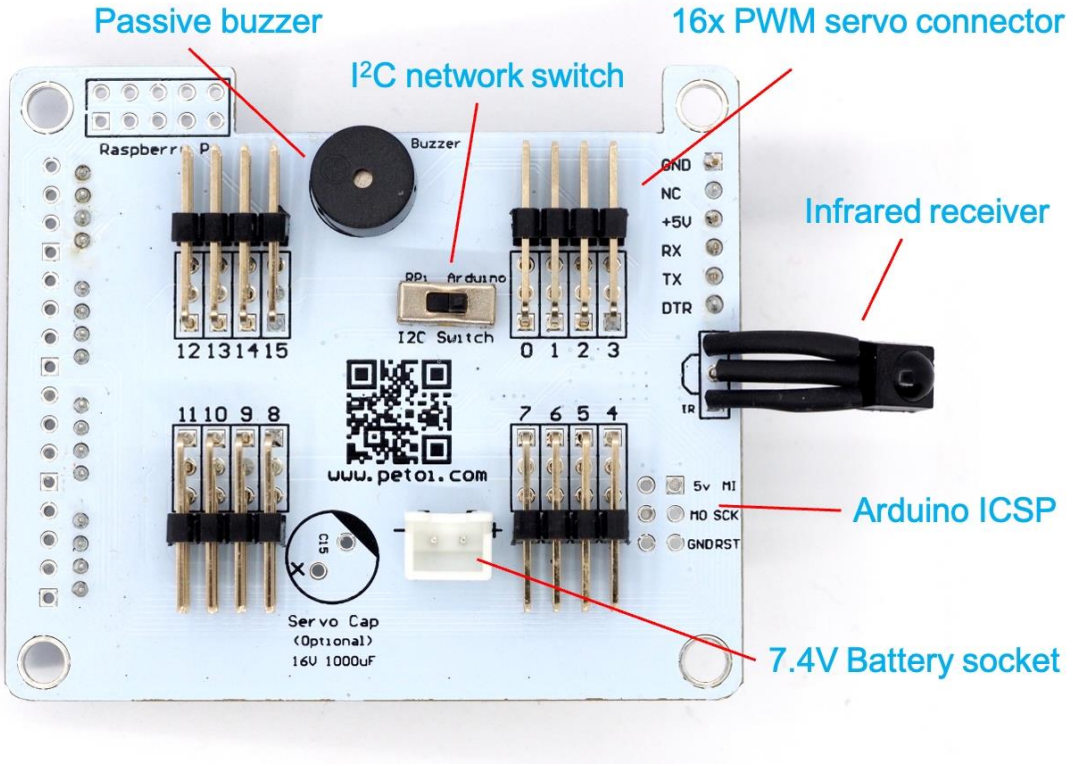
[17] UNIVERSIDAD DE CANTABRIA. M2OS: Sistema Operativo en Tiempo Real para Microcontroladores. Disponible en: https://m2os.unican.es

[18] Petoi. Skill Creation | Petoi Doc Center [en línea]. 2024. Disponible en: https://docs.petoi.com/applications/skill-creation

[19] Petoi, P1S Servo https://www.petoi.com/products/quadruped-robot-dog-bittle-servo-set?variant=49986026373432

Anexo I





https://docs.petoi.com/history/upload-sketch-for-nyboard-software-1.0

Anexo II

```
with Interfaces.C.Strings;

package Arduino.Adafruit_PWM_Servo is
   pragma Elaborate_Body;

   procedure Begin_Adafruit_PWM_Servo
     with Inline_Always;

   procedure Set_PWM_Freq_Adafruit_PWM_Servo (Freq : Interfaces.Unsigned_8)
     with Inline_Always;

   procedure Set_PWM_Adafruit_PWM_Servo (Num : Interfaces.Unsigned_8;
                                         On : Interfaces.Unsigned_16;
                                         Off : Interfaces.Unsigned_16)
     with Inline_Always;


   Adafruit_PWM_Servo_Size_In_Bytes : constant := 9;

   Sizeof_Adafruit_PWM_Servo : Integer with
     Import, Convention => C,
     External_Name => "sizeof_adafruit_pwm_servo";
   --  Defined in 'sizes_arduino_types.cpp' (created by Makefile)
   --  Takes the value of -1 if the library is not included.

private

   type Adafruit_PWM_Servo_Filling is
     array(1 .. Adafruit_PWM_Servo_Size_In_Bytes) of Interfaces.Unsigned_8;

   type Adafruit_PWM_Servo is limited record
      Filling: Adafruit_PWM_Servo_Filling;
   end record;
   pragma Import (CPP, Adafruit_PWM_Servo);
   for Adafruit_PWM_Servo'Size use Adafruit_PWM_Servo_Size_In_Bytes * 8;
   for Adafruit_PWM_Servo'Alignment use Standard'Maximum_Alignment;

   function New_Adafruit_PWM_Servo return Adafruit_PWM_Servo;
   pragma CPP_Constructor (New_Adafruit_PWM_Servo,
                           "_ZN23Adafruit_PWMServoDriverC1Eh");
   ------------------------------------------------
   -- Servo_Controller Adafruit_PWM_Servo object --
   ------------------------------------------------

   Servo_Controller : aliased Adafruit_PWM_Servo := New_Adafruit_PWM_Servo;

end Arduino.Adafruit_PWM_Servo;
```

Archivo arduino-adafruit_pwm_servo.ads

```
with Interfaces.C.Strings;

package body Arduino.Adafruit_PWM_Servo is

   procedure Begin_Adafruit_PWM_Servo(This : access Adafruit_PWM_Servo)
     with
       Import,
       Convention => CPP,
       External_Name => "_ZN23Adafruit_PWMServoDriver5beginEh";

   procedure Set_PWM_Freq_Adafruit_PWM_Servo (This : access Adafruit_PWM_Servo;
                                              Freq : Interfaces.Unsigned_8)
     with
       Import,
       Convention => CPP,
       External_Name => "_ZN23Adafruit_PWMServoDriver22setOscillatorFrequencyEm";

   procedure Set_PWM_Adafruit_PWM_Servo (This : access Adafruit_PWM_Servo;
                                         Num : Interfaces.Unsigned_8;
                                         On : Interfaces.Unsigned_16;
                                         Off : Interfaces.Unsigned_16)
     with
       Import,
       Convention => CPP,
       External_Name => "_ZN23Adafruit_PWMServoDriver6setPWMEhjj";


   procedure Begin_Adafruit_PWM_Servo is
   begin
      Begin_Adafruit_PWM_Servo(Servo_Controller'Access);
   end Begin_Adafruit_PWM_Servo;

   procedure Set_PWM_Freq_Adafruit_PWM_Servo is
   begin
      Set_PWM_Freq_Adafruit_PWM_Servo(Servo_Controller'Access, Freq);
   end Set_PWM_Freq_Adafruit_PWM_Servo;

   procedure Set_PWM_Adafruit_PWM_Servo is
   begin
      Set_PWM_Adafruit_PWM_Servo(Servo_Controller'Access, Num, On, Off);
   end Set_PWM_Adafruit_PWM_Servo;

end Arduino.Adafruit_PWM_Servo;
```

Archivo arduino-adafruit_pwm_servo.adb

```
with Interfaces.C.Strings;

package Arduino.MPU6050 is
   pragma Elaborate_Body;


   procedure Begin_MPU6050
     with Inline_Always;

   procedure Get_Acceletarion_MPU6050 (X : Interfaces.Unsigned_16;
                      Y : Interfaces.Unsigned_16;
                      Z : Interfaces.Unsigned_16)
     with Inline_Always;


   MPU6050_Size_In_Bytes : constant := 9;

   Sizeof_MPU6050 : Integer with
     Import, Convention => C,
     External_Name => "sizeof_mpu6050";
   --  Defined in 'sizes_arduino_types.cpp' (created by Makefile)
   --  Takes the value of -1 if the library is not included.

private

   type MPU6050_Filling is
     array(1 .. MPU6050_Size_In_Bytes) of Interfaces.Unsigned_8;

   type MPU6050 is limited record
      Filling: MPU6050_Filling;
   end record;
   pragma Import (CPP, MPU6050);
   for MPU6050'Size use MPU6050_Size_In_Bytes * 8;
   for MPU6050'Alignment use Standard'Maximum_Alignment;

   function New_MPU6050 return MPU6050;
   pragma CPP_Constructor (New_MPU6050, "_ZN12MPU6050_BaseC1EhPv");

   ------------------------------------------------
   -- MPU_Controller MPU6050 object --
   ------------------------------------------------

   MPU_Controller : aliased MPU6050 := New_MPU6050;

end Arduino.MPU6050;
```

Archivo arduino-mpu6050.ads

```
with Interfaces.C.Strings;

package body Arduino.MPU6050 is

    procedure Begin_MPU6050(This : access MPU6050)
      with
        Import,
        Convention => CPP,
        External_Name => "_ZN12MPU6050_Base10initializeEv";

    procedure Get_Acceletarion_MPU6050 (This : access MPU6050;
                                        X : access Interfaces.Unsigned_16;
                                        Y : access Interfaces.Unsigned_16;
                                        Z : access Interfaces.Unsigned_16)
      with
        Import,
        Convention => CPP,
        External_Name => "_ZN12MPU6050_Base15getAccelerationEPiS0_S0_";


    procedure Begin_MPU6050 is
    begin
      Begin_MPU6050(MPU_Controller'Access);
    end Begin_MPU6050;

    procedure Set_PWM_MPU6050 is
    begin
      Set_PWM_MPU6050(MPU_Controller'Access, X'Access, Y'Access,
Z'Access);
    end Set_PWM_MPU6050;

end Arduino.MPU6050;
```

Archivo arduino-mpu6050.adb