PROYECTO: Monitoreo acústico de niveles de ruido submarino y telemetría satelital para detección de eventos acústicos intensos en aguas de la Plataforma Continental Argentina

Dirección del proyecto: *Igor Prario*
Co-Dirección del proyecto: *Patricio Bos*

Rutinas de configuración y adquisición para placa de audio Behringer

Patricio Bos

División Acústica Submarina
Informe Técnico AS 3/25
Marzo 2025

Rutinas de configuración y adquisición para anemómetro Gill WindSonic.

Marcos Remotti y Patricio Bos

RESUMEN

El presente informe técnico documenta el diseño e implementación de un módulo de software para configurar y controlar una placa de audio Behringer UMC204HD. Este instrumento forma parte del equipamiento de la Estación Autónoma Marítima para el Monitoreo de Ruido Ambiente (EAMMRA), desarrollado por la División Acústica Submarina de la Dirección de Investigación de la Armada (DIIV). El software desarrollado permite la adquisición y registro de señales acústicas analógicas provenientes de hidrofonos de manera automática o manual. Este trabajo es parte del Proyecto "Monitoreo acústico de niveles de ruido submarino y telemetría satelital para detección de eventos acústicos intensos en aguas de la Plataforma Continental Argentina", del programa PIDDEF del Ministerio de Defensa, vinculado al Proyecto "Localización e identificación de fuentes de ruido" de la Armada Argentina, llevado a cabo en la División Acústica Submarina de la Dirección de Investigación de la Armada (DIIV) y la Unidad Ejecutora de Investigación y Desarrollo Estratégicos para la Defensa (UNIDEF), dependiente de CONICET/MinDef.

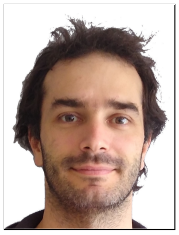
ABSTRACT

This technical report documents the design and implementation of a software module to configure and control a Behringer UMC204HD audio interface. This instrument is part of the equipment of the Autonomous Maritime Station for Ambient Noise Monitoring (EAMMRA), developed by the Submarine Acoustics Division of the Navy Research Directorate (DIIV). The developed software allows for the automatic or manual acquisition and recording of analog acoustic signals from hydrophones. This work is part of the Project "Acoustic monitoring of underwater noise levels and satellite telemetry for detecting intense acoustic events in waters of the Argentine Continental Shelf", from the PIDDEF program of the Ministry of Defense, linked to the Project "Localization and identification of noise sources" of the Argentine Navy, carried out in the Underwater Sound Division of the Argentinian Navy Research Office (DIIV) and the Strategic Research and Development Unit for Defense (UNIDEF), dependent on CONICET/MinDef.

Este trabajo es parte del Proyecto
“Monitoreo acústico de niveles de ruido submarino y telemetría satelital
para detección de eventos acústicos intensos en aguas de la
Plataforma Continental Argentina”,
del programa PIDDEF del Ministerio de Defensa,
vinculado al Proyecto “Localización e identificación de fuentes de ruido”
de la Armada Argentina,
que se lleva a cabo en la División Acústica Submarina
de la Dirección de Investigación de la Armada (DIIV) y
la Unidad Ejecutora de Investigación y Desarrollo Estratégicos
para la Defensa (UNIDEF), dependiente de CONICET/MinDef.

AGRADECIMIENTOS

Los autores desean expresar su agradecimiento a:



Ing. Rui Marques Rojo, investigador RPIDFA que forma parte del equipo de trabajo del Departamento de Propagación Acústica de la Dirección de Investigación de la Armada, por el montaje y puesta a punto de la plataforma Gitea con servicios para el desarrollo de código, que fuera extensamente utilizada en la realización del trabajo que aquí se documenta. Asimismo, se deja constancia de sus valiosas sugerencias para la implementación de código en lenguaje Python.

ÍNDICE

GLOSARIO DE SIGLAS

API	Application Program Interface.
ARA	Armada Argentina.
ASCII	American Standard Code for Information Interchange.
AWG	American Wire Gauge.
CD	Continuous Delivery.
CI	Continuous Integration.
CSV	Comma Separated Values.
DDR	Double Data Rate.
EAMMRA	Estación Autónoma Marítima para el Monitoreo de Ruido Ambiente.
ETX	End of TeXt.
GB	Giga Byte.
GNU	Gnu is Not Unix.
IDE	Integrated Development Environment.
LTS	Long Term Support.
LAN	Local Area Network.
MIL-DTL	Military Detail Specification.
NMEA	National Marine Electronics Association.
PCM	Pulse Code Modulation.
PEP	Python Enhancement Proposal.
RIFF	Resource Interchange File Format.
SSH	Secure SHell.
URL	Uniform Resource Locators.
VPN	Virtual Private Network.
WAV	WAVeform audio file.
YAML	YAML Ain't Markup Language.

LISTA DE FIGURAS

LISTA DE TABLAS

I. INTRODUCCIÓN

El objetivo de este informe técnico es documentar el diseño e implementación de un módulo de software para configurar y controlar una placa de audio Behringer UMC204HD. Este dispositivo forma parte del equipamiento e instrumental de la Estación Autónoma Marítima para el Monitoreo de Ruido Ambiente (EAMMRA) que se desarrolla en la División Acústica Submarina de la Dirección de Investigación de la Armada (DIIV).

La estación EAMMRA es una boya de superficie de diseño específico para la medición de Ruido Ambiente submarino cuya concepción, diseño y fabricación se encuentran documentados en respectivos informes técnicos [Bos *et al.*, 2016], [Ezcurra *et al.*, 2019] y [Cinquini *et al.*, 2019].

El control de la estación en general, y la interacción con la placa en particular, se realizan con una computadora de grado industrial Kaise KBOX-S J6412. Esta computadora pertenece a la categoría de *fanless embedded systems*, que son sistemas de misión específica sin partes móviles y es especialmente adecuada para aplicaciones de funcionamiento autónomo. Cuenta con un procesador Intel Celeron J6412 de cuatro núcleos y memoria DDR4 de 16 GB [Kaise, 2025], lo que le otorga al sistema una razonable capacidad de cómputo para el bajo consumo de energía que requiere la aplicación. En la figura 1 se muestra una vista frontal y trasera de la computadora y se pueden observarlas interfaces y conectores disponibles.

FIG. 1. Vista frontal y trasera de la computadora Kaise KBOX-S J6412.

La computadora de EAMMRA corre un sistema operativo de propósitos generales, Ubuntu Server 22.04 LTS. Ubuntu Server es una distribución libre y gratuita de GNU/Linux que no requiere la compra de licencias para su uso. Este sistema operativo es reconocido por su estabilidad y seguridad, atributos esenciales para aplicaciones críticas en sistemas desatendidos como EAMMRA. Por diseño, Ubuntu Server permite una extensa personalización, con la posibilidad de realizar una instalación con los mínimos componentes necesarios, lo que permite optimizar el rendimiento general del sistema.

El módulo de software para configurar y controlar la placa de audio se implementa en Python, un lenguaje de programación interpretado y multiparadigma de alto nivel que viene integrado por defecto en las distribuciones de Ubuntu. En particular, para este módulo se utiliza Python3 que es la versión del lenguaje recomendada para proyectos nuevos, que no requieren retrocompatibilidad con componentes previos que hayan sido desarrollados en Python2.

I.1. Herramientas de desarrollo

Las herramientas de desarrollo de software que se utilizaron para la implementación del módulo de configuración y control permiten mejorar la calidad del código. Se entiende como código de calidad, aquel que hace lo que se supone que debe hacer, no contienen defectos ni problemas y es fácil de extender con nuevas características.

Las guías de estilo en general se utilizan para definir una forma consistente de escribir código. Si bien el estilo de codificación puede parecer una cuestión de forma, que no afecta el funcionamiento lógico del código, algunas decisiones de estilo pueden evitar errores lógicos frecuentes. Las guías definen convenciones que ayudan a mantener el código fácil de leer, mantener y extender.

Se emplearon herramientas específicas para lenguaje Python, que tienen gran difusión y aceptación en la comunidad de desarrolladores conocidas como *linters*. Estas herramientas permiten analizar el código y verificar el cumplimiento de un conjunto de reglas de mejores prácticas y buscan problemas, tanto de no conformidad con un estilo como errores de lógica. Los defectos de lógica que se buscan son errores de código, código con resultados potencialmente no deseado y patrones de código peligroso. El uso de estas herramientas mejora la legibilidad, la mantenibilidad y la escalabilidad y hace que el código sea menos propenso a errores.

I.1.a. Guías de estilo para código Python - PEP-8 y PEP-257

Para la escritura del código Python del módulo de configuración y control se adopta la guía de estilo *Python Enhancement Proposal* 8 (PEP-8). Su objetivo principal es mejorar la legibilidad y coherencia del código Python y tiene amplia aceptación dentro de la comunidad de desarrolladores de este lenguaje. La guía fue escrita en 2001 por Guido van Rossum, Barry Warsaw y Nick Coghlan, basándose en las mejores prácticas existentes y las recomendaciones de la comunidad [van Rossum *et al.*, 2001]. Esta guía fomenta buenas prácticas y técnicas que pueden ayudar a evitar errores comunes.

La guía cubre varios aspectos de la codificación, que incluyen:

- Formato del código: indentación, uso de espacios en blanco, longitud de las líneas, etc.
- Convenciones de nomenclatura: cómo nombrar variables, funciones, clases, módulos, etc.
- Principios de codificación: recomendaciones sobre cómo escribir expresiones y declaraciones de manera clara.
- Comentarios: cuándo y cómo usar comentarios para mejorar la legibilidad del código.

Asimismo, se utiliza la convención *Python Enhancement Proposal* 257, (PEP-257). Esta convención define el formato y estilo para la documentación, también llamada *Python's docstrings* que aplica a módulos, clases, funciones y métodos [Goodger y van Rossum, 2001]. Adicionalmente, si los *docstring* se escriben en forma consistente, exiten herramientas capaces de generar documentación directamente del código. Esto permite mantener más fácilmente la documentación actualizada dentro del mismo código.

Existen herramientas de desarrollo como pycodestyle y pydocstyle que verifican el estilo y ayudan a asegurar que el código cumpla con las convenciones de PEP-8 y PEP-257, respectivamente. Estas herramientas pertenecen a la categoría de linters de estilo y en algunos casos vienen integrados dentro de otros linters como Flake8 o Pylama.

I.1.b. Style guide enforcement - Flake8

Flake8 es una herramienta que permite imponer y hacer cumplir las directivas de la guía de estilo PEP-8 y pertenece a la familia de programas tipo *lint* que trabajan analizando el código fuente para verificar el cumplimiento de un conjunto definido de reglas. Esta herramienta puede señalar errores de programación, *bugs*, errores de estilo y construcciones sospechosas. Es altamente configurable, y permite a los desarrolladores ajustar las reglas y la severidad de las advertencias según las necesidades específicas de su proyecto [Cordasco, 2016].

Para instalar flake8 para la versión por defecto de Python:

```
python -m pip install flake8
```

Para utilizar flake8, se deben ejecutar los siguientes comando desde una terminal interactiva:

```
flake8 path/to/code/to/check.py
# or
flake8 path/to/code/
```

A su vez, se puede seleccionar una regla específica para ejecutar o ignorar:

```
flake8 --select E123,W503 path/to/code/
# or
flake8 --extend-ignore E203,W234 path/to/code/
```

El listado completo de códigos de error y su significado puede encontrarse en el sitio weboficial de la herramienta, `http://flake8.pycqa.org/en/latest/user/error-codes.html`.

I.1.c. Analizador estático - Pylint

Pylint es una herramienta de análisis estático de código para Python que busca errores de programación, ayuda a hacer cumplir un estándar de codificación y busca "malos olores" en el código (*code smells*). Utiliza diferentes técnicas para analizar el código fuente y puede identificar problemas o patrones de codificación problemáticos que podrían llevar a errores o a un código difícil de mantener o leer [Logilab y contributors to Pylint, 2024].

Las características principales de Pylint incluyen:

- Chequeo de errores: puede detectar errores que podrían hacer que el código falle en tiempo de ejecución, como llamadas a funciones no definidas, uso de variables antes de su definición, etc.
- Estándares de codificación: Pylint puede asegurar de que el código siga un estándar de codificación particular, como PEP-8.
- Refactorización de código: sugiere lugares donde el código podría ser refactorizado para mejorar la legibilidad o la eficiencia.
- Detección de código duplicado: puede identificar bloques de código duplicados que podrían ser simplificados o extraídos en una función común.
- Chequeo de tipos: aunque Python es un lenguaje dinámicamente tipado, Pylint puede realizar algunas comprobaciones de tipos para identificar posibles problemas.

Para instalar pylint para la versión por defecto de Python:

```
python –m pip install pylint
```

Para utilizar pylint, se debe ejecutar el siguiente comando:

```
pylint [options] modules_or_packages
```

Pylint agrega un prefijo a cada una de las áreas problemáticas con una R, C, W, E o F, que significan:

- **R**efactorizar por una violación de la métrica de "buena práctica".
- **C**onvención por violación del estándar de codificación.
- **W**arning (Advertencia) por problemas estilísticos o problemas de programación menores.
- **E**rror por problemas importantes de programación (es decir, muy probablemente un bug).
- **F**atal por errores que impidieron el procesamiento adicional.

La lista completa de mensajes y su significado puede encontrarse en el sitio web oficial de la herramienta, donde se encuentran agrupados por categoría o área problemática. La url es: `https://pylint.pycqa.org/en/latest/user_guide/messages/messages_overview.html`.

I.1.d. Formato automático - Black

Black es una herramienta de formateo de código para Python conocida por su enfoque en la simplicidad y la uniformidad. A menudo se le llama "el formateador de código sin compromisos" debido a su filosofía de tener una sola forma estandarizada y automatizada de formatear el código Python. Esto contrasta con otras herramientas de formateo que pueden permitir una mayor configuración o variaciones en el estilo de codificación [Łukasz Langa y contributors to Black, 2018].

Algunas características clave de Black incluyen:

- Automatización: Black reformatea todo el archivo de código con solo un comando, sin necesidad de ajustes manuales.
- Consistencia: aplica un estilo consistente en todos los proyectos de Python al seguir un conjunto de reglas predefinido, lo que ayuda a mejorar la legibilidad y reducir el tiempo dedicado a discutir sobre estilos de codificación en revisiones de código.
- Integración fácil: puede integrarse fácilmente con editores de texto y entornos de desarrollo integrados (IDEs), así como con sistemas de integración continua/entrega continua (CI/CD).
- Seguridad: está diseñado para realizar cambios en el código que no afecten su comportamiento, lo que lo hace seguro para usar en proyectos grandes y complejos.

Al tener el código un formato unificado entre los distintos desarrolladores, las revisiones de código, especialmente bajo control de versiones, se pueden hacer más rápidamente debido a que los *diffs* entre versiones son lo más pequeños posible. Black es un formateador de código PEP-8 compatible.

Para instalar Black, se debe ejecutar el siguiente comando en una terminal:

```
python pip install black
```

Para utilizar Black, se debe ejecutar el siguiente comando:

```
black modules_or_packages
```

I.1.e. Entorno de Desarrollo Integrado - Visual Studio Code

Visual Studio Code, disponible en `https://code.visualstudio.com/`, es un editor de código fuente de código abierto que soporta múltiples lenguajes de programación. Destaca por su flexibilidad y capacidad de personalización. Permite integrar extensiones que amplían sus funcionalidades. Entre sus características se encuentran el soporte para depuración integrada, control de versiones con Git y herramientas de autocompletado. También ofrece una interfaz de usuario intuitiva y un sistema de gestión de proyectos eficiente. VSCode es compatible con Windows, macOS y GNU/Linux.

Existen múltiples extensiones disponibles para Python. Entre las más destacadas se encuentran el plugin `Python`, que proporciona soporte para depuración, ejecución de código, y autocompletado de código Python, y `Pylance`, que ofrece características adicionales como análisis estático de código y autocompletado inteligente.

La extensión Remote - SSH de Visual Studio Code permite utilizar cualquier máquina remota con un servidor SSH como entorno de desarrollo. Esto facilita el desarrollo y la resolución de problemas en una amplia variedad de situaciones. Mediante esta extensión, es posible desarrollar en el mismo sistema operativo al que se desplegará el software o emplear hardware más grande, rápido o especializado que el de la máquina local.

Una de las ventajas de esta extensión es la capacidad de cambiar rápidamente entre diferentes entornos de desarrollo remotos, lo que permite realizar actualizaciones de manera segura sin riesgo de afectar la máquina local. Además, proporciona acceso a un entorno de desarrollo ya existente desde varias máquinas o ubicaciones, lo que aumenta la flexibilidad del proceso de desarrollo.

Esta extensión resulta útil también para depurar aplicaciones que se ejecutan en otros lugares. No es necesario tener el código fuente en la máquina local para aprovechar estas ventajas, ya que la extensión ejecuta comandos y otras extensiones directamente en la máquina remota. Es posible abrir cualquier carpeta en la máquina remota y trabajar con ella de la misma forma que si estuviera en la máquina local.

II. PLACA DE AUDIO BEHRINGER UMC204HD

La Behringer UMC204HD es una interfaz de audio de alta calidad diseñada para la captura y procesamiento de señales de audio. Esta interfaz es compatible con sistemas operativos modernos Windows y GNU/Linux y se conecta a través de un puerto USB 2.0. Su capacidad para trabajar con alta resolución vertical y elevadas tasas de muestreo la convierte en una herramienta útil para una variedad de aplicaciones profesionales en el ámbito del análisis de señales. En la figura 2 se puede apreciar una vista frontal y trasera de la interfaz de audio UMC204HD.

La Behringer UMC204HD ofrece una resolución vertical de 24 bits, lo que permite una captura precisa y detallada de las señales de entrada. Esta alta resolución es esencial para la adquisición de datos en aplicaciones que requieren una representación fiel de las variaciones más pequeñas en la señal, como es el caso de señales adquieridas con hidrófonos.

La interfaz tiene una tasa de muestreo de hasta 192 kHz, lo que le permite capturar señales con una adecuada resolución temporal. Esta alta tasa de muestreo es crucial para evitar el fonómeno de *aliasing* y para asegurar que las señales con contenido espectral en altas frecuencias, como las señales provenientes de los hidrófonos, se capturen adecuadamente. Asimismo, la interface UMC204HD se caracteriza por tener bajo nivel de ruido propio.

FIG. 2. Vista frontal y trasera de la interface de audio Behringer UMC204HD.

III. DISEÑO E IMPLEMENTACIÓN

III.1. Descripción general del módulo

El módulo `BehringerHandler` fue desarrollado en Python con el objetivo de controlar una interfaz de audio USB marca Behringer de forma programática. Permite su inicialización, apertura de flujos de entrada, grabación asincrónica y gestión completa de recursos. Está pensado para ser utilizado en sistemas de adquisición acústica en tiempo real, y su diseño modular facilita su integración en arquitecturas mayores, como estaciones de monitoreo o sistemas embebidos.

El funcionamiento interno del módulo se apoya en varias bibliotecas estándar y externas. Utiliza `pyaudio` para el acceso de bajo nivel al hardware de audio, `wave` para la escritura de los datos capturados en formato WAV, `threading` para realizar la grabación en paralelo al hilo principal y `queue` para sincronizar los datos entre procesos. Además, se emplean `logging` para la trazabilidad de eventos, `datetime` para la generación de nombres de archivo basados en marca temporal y `os` para la gestión del entorno de ejecución y rutas de archivo. Esta combinación de módulos permite una operación robusta, escalable y fácilmente depurable.

La biblioteca `logging` de Python se utiliza para generar registros estructurados durante la ejecución del código, y permite monitorear el estado del sistema, detectar errores y realizar trazabilidad de eventos. En el módulo `BehringerHandler`, se implementa un sistema de log dual que escribe tanto en consola como en un archivo de texto (`behringer_handler.log`). Este sistema utiliza formatos personalizados con marcas de tiempo y niveles de severidad (INFO, WARNING, ERROR), lo que facilita el diagnóstico en tiempo real y la revisión posterior del funcionamiento del sistema. La inclusión de `logging` mejora la mantenibilidad y confiabilidad del software, especialmente en entornos donde no hay interacción directa con el usuario.

La biblioteca `PyAudio` proporciona una interfaz en Python para interactuar con PortAudio, una biblioteca de código abierto para la captura y reproducción de audio en tiempo real. Este paquete permite abrir flujos de entrada o salida de audio con parámetros configurables como tasa de muestreo, formato, número de canales y tamaño de buffer. En el caso de `BehringerHandler`, `PyAudio` es utilizado para detectar dispositivos de audio disponibles, abrir un flujo de entrada en formato `paInt24` y capturar señales a alta resolución (192 kHz, 2 canales). Gracias a su capacidad para operar en modo no bloqueante mediante callbacks, `PyAudio` permite implementar grabaciones asincrónicas eficientes, fundamentales para aplicaciones de adquisición continua o en tiempo real.

La biblioteca estándar `wave` de Python permite la manipulación de archivos de audio en formato WAV (RIFF). Proporciona una interfaz sencilla para la escritura y lectura de audio en formato PCM (Pulse Code Modulation), y permite definir parámetros como la cantidad de canales, la tasa de muestreo y la resolución en bits. En el módulo `BehringerHandler`, `wave` se utiliza para almacenar las grabaciones capturadas por el stream de PyAudio en archivos de alta calidad de 24 bits. La escritura se realiza en tiempo real desde un hilo separado, para asegurar la integridad de los datos y su disponibilidad inmediata en disco para posteriores análisis.

La biblioteca `threading` proporciona soporte para la ejecución concurrente de código mediante la creación y gestión de hilos de ejecución. En `BehringerHandler`, se utiliza para realizar la escritura de audio en disco de forma asincrónica, separando la captura en tiempo real del procesamiento de los datos. Esto permite que el flujo principal de adquisición no se vea interrumpido por operaciones de entrada/salida, lo que mejora el rendimiento y evita pérdidas de datos. El uso de `threading` resulta esencial para garantizar grabaciones continuas y confiables, especialmente en sistemas que requieren alta tasa de muestreo o baja latencia.

La biblioteca `queue` proporciona una estructura de datos segura para comunicación entre hilos en entornos multithreading. En el módulo `BehringerHandler`, se utiliza una instancia de `Queue` para almacenar de forma temporal los bloques de audio recibidos por el callback del stream de PyAudio. Esta cola permite desacoplar la adquisición de datos (en tiempo real) de la escritura en disco, evitando pérdidas o bloqueos por operaciones lentas de entrada/salida. El uso de `queue` garantiza una sincronización segura y eficiente entre el hilo de captura y el hilo de grabación.

La biblioteca estándar `datetime` permite manipular fechas y horas de manera precisa y flexible. En el módulo `BehringerHandler`, se utiliza para generar marcas de tiempo en el momento de iniciar una grabación, lo que permite construir nombres de archivo únicos con el formato `YYYYMMDD_HHMMSS`. Esta práctica facilita la organización de las grabaciones, evita sobreescrituras accidentales y proporciona una trazabilidad temporal inmediata. El uso de `datetime` también permite registrar eventos en los logs con exactitud temporal.

La biblioteca estándar `os` proporciona funciones para interactuar con el sistema operativo de forma portable. En el módulo `BehringerHandler`, se utiliza para obtener la ruta absoluta del script en ejecución, crear subdirectorios (como `recordings` para guardar archivos de audio) y construir rutas de forma segura e independiente del sistema operativo. El uso de `os` es esencial para garantizar la correcta gestión de archivos y configuraciones en distintos entornos de ejecución.

III.2. Dependencias y gestión del entorno

La lista completa de dependencias necesarias para ejecutar el módulo `BehringerHandler` se encuentra documentada en el archivo `requirements.txt`. Este archivo fue generado utilizando el comando `pip freeze >requirements.txt` desde un entorno virtual de Python configurado específicamente para este proyecto. La utilización de entornos virtuales permite aislar las versiones de los paquetes, asegurando la reproducibilidad del entorno de ejecución.

Para reconstruir el entorno en una nueva máquina o entorno limpio, se recomienda crear un entorno virtual (por ejemplo, mediante `python -m venv venv`) y luego instalar todas las dependencias mediante el comando `pip install -r requirements.txt`. Esto garantiza que las versiones exactas utilizadas durante el desarrollo serán replicadas, y permite reducir los errores por incompatibilidades entre bibliotecas.

A continuación, se presenta la lista de dependencias incluidas en `requirements.txt`:

```
contourpy==1.3.1
cycler==0.12.1
fonttools==4.56.0
iniconfig==2.1.0
kiwisolver==1.4.8
matplotlib==3.10.1
numpy==2.2.3
packaging==24.2
pillow==11.1.0
pluggy==1.5.0
PyAudio==0.2.14
pyparsing==3.2.1
pytest==8.3.5
python-dateutil==2.9.0.post0
schedule==1.2.2
six==1.17.0
```

III.3. Arquitectura del módulo

El diseño está centrado en la clase `BehringerHandler`, que actúa como fachada para todas las operaciones necesarias. La arquitectura sigue una lógica secuencial:

1. Inicialización del entorno (`init`): escanea los dispositivos de audio y selecciona automáticamente el que corresponde a Behringer.
2. Apertura del flujo de entrada (`open`).
3. Grabación asincrónica (`record` y `_write_audio`) mediante un hilo dedicado y una cola de datos.
4. Cierre y liberación de recursos (`stop_recording`, `close`, `deinit`).

En la figura 3 se presenta un diagrama de flujo general con los bloques principales y la secuencia lógica de ejecución del módulo Beheringer_Handler.

Durante la ejecución, los eventos relevantes se registran tanto en consola como el archivo con extensión .log.

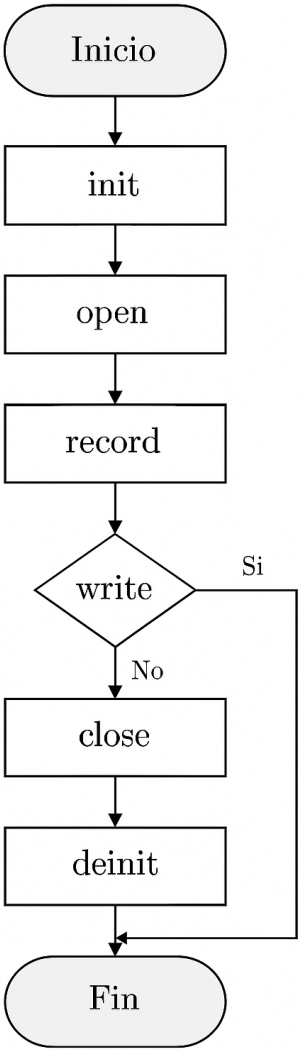


Figura 1: Diagrama de flujo de funcionamiento del módulo

FIG. 3. Diagrama de flujo de funcionamiento del módulo `BehringerHandler`

III.4. Uso del módulo

El flujo de uso del módulo es el siguiente:

```
from behringer_handler import BehringerHandler

handler = BehringerHandler()
handler.init()
handler.record(duration=10)  # graba 10 segundos de audio
handler.stop_recording()
handler.deinit()
```

III.5. Descripción de la clase y sus métodos

`BehringerHandler` extiende de `BaseHandler` e implementa los métodos clave para el manejo del dispositivo. La utilización de una clase base común responde a una estrategia de diseño orientada a la modularidad y escalabilidad del sistema EAMRRA, donde múltiples módulos de adquisición o control comparten una estructura funcional similar.

La clase `BaseHandler` actúa como plantilla abstracta, y define una interfaz mínima que obliga a los módulos derivados a implementar ciertos métodos esenciales, como `init`, `deinit`, `open` y `close`. Esta decisión asegura una simetría en la arquitectura del sistema, facilita el mantenimiento y permite la integración homogénea de nuevos módulos en el futuro. Además, permite aplicar patrones de programación como la inversión de dependencias, delegando a cada módulo el detalle de implementación mientras se conserva una estructura común y predecible.

A continuación se describen los métodos principales de la clase `BehringerHandler`:

`__init__`

Constructor de la clase. Inicializa atributos, crea el logger y configura los manejadores de archivo y consola.

`init()`

Escanea los dispositivos de audio disponibles con PyAudio. Si encuentra uno con el nombre `Behringer` o que incluya `USB` y tenga canales de entrada, lo selecciona y guarda su índice.

`open()`

Abre un stream de entrada en modo no bloqueante para el dispositivo detectado, utilizando formato `paInt24`, dos canales, y frecuencia de muestreo de 192 kHz. Registra advertencias si el dispositivo no está inicializado.

`_callback(in_data, frame_count, time_info, status)`

Función callback del stream. Si la grabación está activa, coloca los datos de entrada en una cola de frames. Si no, retorna `paComplete`.

`record(duration)`

Inicia una grabación de duración determinada (en segundos). Crea un hilo que ejecuta `_write_audio`, abre el stream e inicia la adquisición. El archivo de salida se guarda con nombre basado en fecha y hora en la carpeta `recordings`.

`_write_audio()`

Hilo que extrae datos de la cola y los escribe dinámicamente en un archivo WAV. Utiliza `wave` para manejar el formato y guarda las grabaciones en disco hasta que se completa el tiempo solicitado.

`stop_recording()`

Detiene la grabación, cierra el hilo de escritura y registra el evento.

`deinit()`

Libera el objeto PyAudio, permite reinicializar el sistema si se vuelve a ejecutar `init()`.

`close()`

Detiene y cierra el stream de audio si está activo. Libera la referencia.

III.6. Diagrama de flujo del módulo

IV. PRUEBAS Y ENSAYOS

V. CONCLUSIONES

REFERENCIAS

Bos, P., Cinquini, M., Prario, I., y Blanc, S. (2016). EAMMRA: Ingeniería conceptual. INF. TEC. AS 03/16, DAS, DIIV.

Cinquini, M., Bos, P., Prario, I., y Rojo, R. M. (2019). Integración de subsistemas para el prototipo de eammra. INF. TEC. AS 02/19, DAS, DIIV.

Cordasco, I. S. (2016). Flake8: Your tool for style guide enforcement. `https://flake8.pycqa.org/en/latest/#`. Accedido: 05/03/2024.

Ezcurra, H., Prario, I., Bos, P., Cinquini, M., y Blanc, S. (2019). Diseño de una boya prototipo para medición de nivel de ruido submarino en zona costera. INF. TEC. AS 01/19, DAS, DIIV.

Gill Instruments Ltd (2019). *WindSonic Ultrasonic Anemometer User Manual*. Gill Instruments Ltd. Doc No 1405-PS-0019.

Goodger, D., y van Rossum, G. (2001). Python enhancement proposals 257 – docstring conventions. `https://peps.python.org/pep-0257/`. Accedido: 04/03/2024.

IEC (2020). *IEC 60529 Degrees of protection provided by enclosures (IP Code)*. International Electrotechnical Commission, Geneva, Switzerland, 2.2 ed. Standard.

Kaise (2025). Kbox-s j6412. `https://www.tempelgrouplatam.com`. Folleto publicado por TEMPEL GROUP.

Logilab, y contributors to Pylint (2024). Pylint. `https://pylint.readthedocs.io/en/stable/`. Accedido: 05/03/2024.

van Rossum, G., Warsaw, B., y Coghlan, A. (2001). Pep 8 – style guide for python code. `https://peps.python.org/pep-0008/`. Accedido: 04/03/2024.

Łukasz Langa, y contributors to Black (2018). The uncompromising code formatter. `https://black.readthedocs.io/en/stable/`. Accedido: 05/03/2024.