SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Morelia

"José María Morelos y Pavón"

División de Estudios de Profesionales
Departamento de Ingeniería Electrónica

titulación integral por tesis

"Desarrollo de un sistema de adqusición de datos basado en Linux embebido para la medición de concentraciones de gases en reacciones sólido gas"

que para obtener el título de:

Ingeniero en Electrónica

presenta:
Edwars Sayeth Rodríguez Martínez

Asesor:
Gerardo Marx Chávez Campos

[5 de marzo de 2025 at 12:00 – itmorelia version 1.0]

Mesa de revisión:
Gerardo Marx Chávez Campos
James Clerk Maxwell
Piere Simon Laplace
Jean-Baptiste Joseph Fourier

Localidad:
Morelia, Michoacán, México

Impresa:
Mes 2025

Ohana significa familia.
Y tu familia nunca te abandona ni te olvida.

— Lilo & Stitch –

– Dedicado en memoria de George Simon Ohm –
1789 – 1854

Prólogo

Los documentos de tesis son un legado escrito que prevalece por los años. Este documento, dependiendo de la institución que lo emite, puede variar en la forma que se presenta. En ese mismo sentido, la estructura ha sido definida en los lineamientos emitidos por la Dirección General de Educación Superior Tecnológica (DGEST). Sin embargo, la falta de actualización, la complejidad de manejar documentos de gran capacidad por procesadores como Microsoft Word y una mayor uniformidad de tesis en la División de Estudios de Posgrado e Investigación (DEPI). Han propiciado que el Comité Institucional de Posgrado e Investigación emitan esta primera versión de un manual/formato para la estructura de documentos como tesis, tesinas o disertación.

En esta guía/formato se presentan e identifican los elementos y tipografía basados en el ISO-7144 y la DGEST, así como su implementación en lenguaje LaTeX, éste a su vez funciona como una guía para que el autor pueda redactar y estructurar adecuadamente las partes del documento.

La plantilla está configurada para ejecutarse con cualquiera de las distribuciones libres de LaTeX como MiKTeX o TeXLive, además ha sido probada en **Overleaf**, **TeXPad** y muestra absoluta compatibilidad con **Mendeley** y **Plot.ly**; el código está ampliamente basado en la plantilla **"Classic Thesis Template"** del autor **Andre Miede**.

– Gerardo Marx Chávez Campos–

Abstract

Add your abstract here ...

Resúmen

Agregar resumen aquí ...

Índice general

Índice de figuras

Índice de tablas

Sección de Códigos

Capítulo 1

Introducción

1.1 Semblanza del problema

El acero ha impulsado el desarrollo de la sociedad desde que comenzó a utilizarse, y seguirá siendo la columna vertebral y el facilitador de la evolución y el progreso[1]. Actualmente el acero es un material indispensable, este material es ampliamente utilizado en diversas áreas, y es considerado el más importante dentro de la ingeniería y la construcción ya que ningún otro reúne sus características, como resistencia, plasticidad y versatilidad, además de poder ser reciclado fácilmente sin perder sus propiedades[2][3]. Tan solo en el 2019 se emplearon 1,767.5 millones de toneladas de acero en el mundo, utilizándose un 52 % en edificios e infraestructuras, 16 % en herramientas y maquinaria, 12 % en automóviles, 5 % en otros medios de transporte, 10 % en productos de metal, 3 % en equipo eléctrico y 2 % en artículos domésticos[1].

El amplio uso del acero se debe también a la gran variedad de aceros que hay. En la actualidad existen más de 3,500 grados de acero con propiedades físicas, químicas y ambientales distintas[2]. Esta gran variedad de aceros ha sido posible gracias al estudio y desarrollo de los tratamientos que se aplican a este material para modificar sus propiedades, dentro de éstos, mayormente los tratamientos térmicos.

Durante los tratamientos térmicos el acero es sometido a curvas de temperatura, las cuales pueden llegar a temperaturas muy altas. Inevitablemente, la mayoría de los metales sufren de oxidación a altas temperaturas. En el acero surge un fenómeno denominado *mill scaling* durante estos tratamientos, el cuál consiste en la formación de una delgada capa de óxido en la superficie de éste. Esta delgada capa se agrieta con facilidad, pasando de ser una barrera protectora a una vulnerabilidad corrosible, por lo que debe ser removida para evitar el deterioro del material. [4][5].

Este fenómeno representa un gran inconveniente para la industria del acero ya que ocasiona el desperdicio de al rededor 1 % al 2 % de este material, el cuál no es nada despreciable al tratarse de la inmensa producción a nivel mundial. A esta perdida de material también se agregan los recursos que se requieren durante la labor de remover y reciclar esta capa de óxido, por lo que resulta muy conveniente aminorar este fenómeno[5].

El *mill scaling* depende de tres factores principales: la química del acero, la temperatura y los gases de la atmósfera. Ya que tanto la química del acero como la temperatura determinan las propiedades deseadas del material, la opción más viable es variar los gases dentro de la atmósfera con el propósito de aminorar este fenómeno[5], por este motivo es necesario estudiar esta relación existente entre los gases presentes durante los tratamientos térmicos y el *mill scaling*.

Para este estudio se necesitan realizar diferentes pruebas donde se registren los comportamiento de estas variables. Durante estas pruebas se debe someter a la pieza de acero a un tratamiento térmico experimental, el cual debe realizarse siguiendo una curva de temperatura determinada. Dichas curvas suelen tomar bastante tiempo (por efectos de la inercia térmica), lo que hace que estas pruebas sean demasiado y tardadas. En estas pruebas también se debe de ajustar la frecuencia con la cual se toman las mediciones, adecuándola a la dinámica de las variables medidas, así como la duración de la adquisición de estos datos dependiendo de la duración de la prueba. También son necesarios varios tipos de sensores para poder medir las diferentes condiciones atmosféricas (como temperatura, humedad, presión y concentraciones dentro de la atmósfera), los cuales deben de ser compatibles con la plataforma de adquisición de datos, con la cuál se registrarán estas mediciones.

Los sistemas de adquisición de datos (*DAQ*, por *Data Acquisition*) son comúnmente utilizados en la realización de este tipo de pruebas, los cuales ayudan en su automatización capturando las mediciones de los sensores, e inclusive pudiendo utilizar actuadores para controlar acciones necesarias en ciertas pruebas.

Lamentablemente, los sistemas *DAQ* comerciales, al ser de entorno de desarrollo cerrado, están limitados a las funciones proporcionadas por el fabricante dentro del mismo entorno. Esto es una gran desventaja para las pruebas, ya que difícilmente se podrían tener funciones útiles de forma nativa, como el monitoreo remoto, que resultaría de gran utilidad al tratarse de pruebas muy largas. De la misma forma, al utilizar estos sistemas comerciales, el *hardware* que se puede emplear (como sensores y algunos actuadores) se reduce a una gama limitada con los cuales el sistema *DAQ* es compatible. Esta limitada selección de sensores podría no ser suficiente, o los sensores podrían no ser útiles para el propósito específico de las pruebas. Por otra parte, un entorno de desarrollo no permitiría ajustes a un bajo nivel del funcionamiento del sistema *DAQ*, lo que limita enormemente en una forma general la flexibilidad del sistema para ajustarse a las necesidades específicas que surjan en la realización de las pruebas. Además, utilizar sistemas de adquisición existentes en el mercado también tiene algunas otras desventajas, como requerir un equipo de computo potente, poder utilizarlos sólo con el sistema operativo *Windows*, o tener que adquirir de licencias, *hardware* y funciones adicionales costosas.

1.2 Revisión del estado del arte

Desde el auge de los microcontroladores, estos han sido utilizados para varias aplicaciones que se han ido diversificando a través de los años hasta estos tiempos. Hoy en día, los microcontroladores son una parte fundamental de la electrónica, realizando una amplia variedad de tareas de forma simple y fácilmente modificables.

Con el surgimiento de los procesadores ARM (Advanced Risc Machine), los microcontroladores se vuelven herramientas más poderosas, pudiendo realizar tareas más complejas, o correr sistemas operativos que permiten realizar varias tareas simultáneamente.

Dentro del campo de la adquisición de datos se han implementado diferentes tipos sistemas con diferentes microcontroladores (siendo la tarjeta *Beaglebone* muy popular por su configurabilidad y capacidades). En [6] se utiliza un procesador ARM9 con RTLinux para implementar un *Sistema Inter-*

activo de Adquisición de Datos y Control en linea, utilizando HTML para diseñar la página web. Este sistema permite obtener mediciones de diferentes variables y después mostrarlas en una página web que puede ser accesada remotamente a través de un navegador web. El sistema puede ser ampliamente utilizado en campos como energía eléctrica, petroleo, química, metalurgia, acero, transporte, industrias Electrónica y Eléctrica y automoción entre otras.

En [7] diseñan e implementan un sistema de adquisición de datos embebido basado en Linux para Redes Inteligentes, donde se utiliza una tarjeta *on-board computer BeagleBone*, que es una plataforma de desarrollo embebido desarrollado por *Texas Instruments* con un microprocesador ARM Cortex-A8 y un sistema operativo Ubuntu 12.04. El sistema es capaz de realizar la adquisición, recibiendo mediciones de un medidor inteligente a través del puerto Ethernet, transmisión y almacenamiento de la información a través de la red, creando una base de datos con esa información. Además se encarga del control remoto mientras mantiene la operación de la red estable, mostrando las capacidades de la tarjeta BeagleBone.

En [8] analizan el uso de las *Unidades de Tiempo Real Programables* (PRU, por sus siglas en inglés, *Programable Real-Time Unit*) de la tarjeta *Beaglebone Black*, que es una versión mejorada de la tarjeta *BeagleBone*. El uso de las PRU es estudiado para realizar tareas en tiempo real y es comparado con el rendimiento obtenido con el procesador ARM y con el kernel de tiempo real Xenomai. Los resultados indican que el tiempo promedio de obtención de mediciones utilizando el procesador ARM es de 1,300 microsegundos, mientras que utilizando *kernel* Xenomai es de 1,000 microsegundos y con el uso de las PRU el tiempo promedio de obtención de mediciones es de 10 microsegundos, siendo este último método el más rápido en una relación de aproximadamente cien veces, lo cuál muestra el potencial de la tarjeta Beaglebone Black para realizar tareas en tiempo real con sus PRU mientras puede correr un sistema operativo en su procesador ARM para realizar otro tipo de tareas.

En [9] utilizan una Beaglebone Black para implementar un sistema de adquisición de datos en tiempo real, con una alta precisión y velocidad, siendo además configurable y de de bajo costo. Este sistema utiliza el sistema operativo Debian como sistema operativo, empleando una de sus PRU para controlar la frecuencia de muestreo del convertidor analógico-digital. Con esta configuración se logra obtener muestras con los dos canales del convertidor, con una resolución de 24 bits a una taza de 130,208 muestras por segundo. En dicho trabajo el sistema es utilizado con un hidrófono para para crear satisfactoriamente un sistema de adquisición de datos acústico submarino, pudiendo equipararse con opciones disponibles en el mercado, añadiendo además ventajas como su versatilidad y su bajo costo.

En [10], emplean la PRU de la *Beaglebone Black* para operar el *ADC* interno de la tarjeta a 5000 muestras por segundo, con el objetivo de detectar fugas de fondo en tuberías metálicas por medio de métodos vibro-acústicos. Mientras que la PRU0 se encarga de operar el *ADC*, la PRU1 va obteniendo las muestras de la FIFO del *ADC*. Las PRU fueron programadas en ensamblador para tener un mayor control sobre los tiempos de operación. Cabe mencionar que en este mismo trabajo se desarrolla una herramienta que facilita la utilización de las PRU

Ya que el lenguaje ensamblador puede ser complicado para usuarios no adiestrados, se utiliza un sistema de alto nivel que gestione las aplicaciones de las PRU creado por ellos mismos.

Para gestionar la aplicación principal de la adquisición de datos se utiliza un programa escrito en Python para facilitar el código, haciendo uso de la librería PyPRUSS disponible en Internet que se encarga de cargar el programa a la PRU y comunicarse con estos a través de interrupciones.

El desempeño de este sistema fue comparado con la función `time.sleep()` de Python y un temporizador interno del SoC (System on Chip) para accionar el *ADC*, mostrando que, mientras haciendo uso de las PRU el error es pequeño y no aumenta con la frecuencia de muestreo, con los otros métodos

el error es mucho mayor y aumenta de forma exponencial al aumentar la frecuencia de muestreo.

En [11] desarrollan un sistema de adquisición de datos en tiempo real, open source y acceso remoto vía explorador web utilizando una tarjeta *Beaglebone Black* con sistema operativo Debian para capturar y registrar la temperatura y los eventos de un sistema electrónico. Este sistema tiene un reloj de tiempo real conectado por comunicación I^2C, con el cual se obtiene la fecha y hora exacta e implementa un servidor web embebido con una base de datos creada con MySQL. La información generada del sistema eléctrico monitoreado también puede ser respaldada conectando una tarjeta microSD. Es sistema es de bajo costo y pequeño, y puede seguir funcionando después de un corte de energía gracias a que incorpora una batería.

Por otra parte, en el área de monitoreo (o muestreo) de concentraciones de gases, en [12] se presentan el desarrollo de un sistema de monitoreo remoto en tiempo real de gases peligrosos (bióxido de carbono y monóxido de carbono) presentes en el ambiente, basado en la web, utilizando el software de *National Instruments LabVIEW* con una tarjeta de adquisición de datos basada en un microcontrolador de bajo costo (Arduino Uno). En este trabajo, los datos son mostrados en un panel dentro del software LabVIEW y continuamente almacenados en una base de datos en Excel. Para acceder de forma remota con un navegador web se hace uso de la herramienta de publicación web de LabVIEW.

item

Lista de tareas pendientes

Todas estas características muestran el potencial de la tarjeta *Beaglebone Black* y las diferentes funcionalidades que pueden ser agregadas a ésta, aumentando su versatilidad para diferentes sistemas de adquisición de datos con diferentes necesidades.

1.3 Solución propuesta

En base a lo anterior, con el propósito de obtener la información de las condiciones atmosféricas en el proceso de descaburación (*mill scaling*), dentro de un reactor controlado para su posterior estudio, se propone desarrollar un sistema de adquisición de datos de bajo costo basado en herramientas *open-source*, utilizando la plataforma de desarrollo *Beaglebone*, sentando bases para sistemas con mayor sofisticación.

1.4 Objetivos

1.4.1 Objetivo general

Implementar un sistema de adquisición de datos en un sistema embebido con SO Linux para realizar mediciones de las concentraciones de CO, CO_2 y O_2, así como temperatura en una atmósfera controlada para su posterior análisis.

1.4.2 Objetivos particulares

- Diseñar e implementar un circuito de protección y aislamiento para los puertos UART
- Desarrollar funciones de código para la transmisión y recepción de datos para la comunicación UART en la tarjeta Beaglebone Black
- Comunicar con los sensores vía UART utilizando la tarjeta Beaglebone Black y el circuito de protección y aislamiento para los puertos UART

- Desarrollar funciones de código para la interacción con los sensores en la tarjeta Beaglebone Black
- Desarrollar funciones de código para el registro de mediciones de los sensores en la tarjeta Beaglebone Black
- Desarrollar funciones de código para realizar peticiones de mediciones a los sensores a una frecuencia y durante un tiempo determinado por un usuario
- Verificar el funcionamiento del sistema midiendo concentraciones de gases en una prueba controlada

1.5 Justificación

Para la elaboración de este trabajo se desarrollaran herramientas que aportarán en el estudio de las reacciones sólido-gas en los tratamientos térmicos, esto permitirá conformar un sistema de adquisición de datos que se ajuste de la forma deseada a las necesidades de las pruebas y experimentos de estas reacciones, con lo cuál podrán optimizarse en cierta medida los tratamientos térmicos logrando reducir el *milling* y con ello el desperdicio de material.

Optando por la utilización de herramientas *open-source* se agilizará y potenciará el desarrollo de este sistema, agregando las funcionalidades requeridas, además de eliminar la dependencia de sistemas operativos, entornos de desarrollo y DAQ comerciales con funcionalidades limitadas.

Capítulo 2

Marco teórico

En este capítulo se presenta una recopilación de investigaciones previas e información teórica relacionada con el sistema realizado en este trabajo. Los temas que se abordan en este marco teórico son: adquisición de datos y sus partes básicas, una breve descripción de los sistemas embebidos, así como Linux embebido, el sistema operativo *Debian*, la *single-board computer Beaglebone Black*, el protocolo *Secure Shell*, el entorno de desarrollo *Cloud9*, y las librerías de C `termios.h` y `time.h`. También se explica que es un dispositivo UART y se muestra la información general de: los sensores de gas utilizados en este sistema y de un sistema de asimiento digital basado en el CI *ISOW7842*.

2.1 Adquisición de datos

La adquisición de datos es un proceso en el cuál se obtiene información de uno o varios fenómenos físicos a través del tiempo. Dichas variables son medidas con sensores, los cuales transforman una magnitud de una variable física en un potencial eléctrico. Las mediciones pueden ser obtenidas a través de un sistema de adquisición y almacenadas en algún otro dispositivo. Los sistemas de adquisición de datos más comunes, como el mostrado en la Figura 2.1, se componen de tres partes principales: sensores, dispositivo *DAQ* (por sus siglas en inglés, Data Acquisition) y un equipo de computo[13].

Figura 2.1: Diagrama de las partes de un sistema de adquisición de datos[13].

2.1.1 Sensores

Los sensores son transductores que transforman una variable física en una señal eléctrica que puede ser voltaje, corriente, resistencia o algún otro atributo electrónico. Esta señal comúnmente requiere un acondicionamiento, donde se puede filtrar el ruido, establecer los valores mínimos y máximos de ésta, entre otras cosas. Una vez que la señal sea apta, ésta podrá ser leída e interpretada por un dispositivo electrónico de manera segura y adecuada[13].

2.1.2 Dispositivo *DAQ*

La función del dispositivo *DAQ* es hacer de interfaz entre la computadora y los sensores, obteniendo las mediciones de las variables físicas y digitalizándolas para que la computadora pueda interpretarlas. Este dispositivo *DAQ* puede componerse de tres partes: el circuito de acondicionamiento de señales, un *ADC* (acrónimo de analogue-to-digital converter), y un bus de comunicación con la computadora, que puede ser *Ethernet*, *Wi-Fi*, *USB*, *PCI* o algún otro[13][14].

El circuito de acondicionamiento de señales toma la señal eléctrica proveniente del sensor y la modifica para que la señal sea segura y libre de ruido y apta para ser medida por el convertidor *ADC*. Por su parte, el convertidor *ADC* toma la señal del sensor acondicionada y crea una representación digital de ésta en un instante en el tiempo. Esta operación es repetida periódicamente para obtener una representación digital de una señal analógica cambiante a través del tiempo. La función del bus de comunicación es ser la interfaz entre el dispositivo *DAQ* y la computadora para poder enviar instrucciones y recibir las mediciones realizadas[13].

De esa forma los elementos de un sistema de adquisición de datos interactúan, para así registrar y observar el mundo físico. Ésta información permite analizar el comportamiento de los fenómenos, lo que nos ayuda a tomar decisiones informadas.

2.1.3 Computadora y software

La computadora con el software se encarga de controlar la operación del dispositivo *DAQ* además de procesar, visualizar y almacenar la información de las variables físicas obtenidas. El software facilita la interacción entre la computadora y el usuario para poder obtener, registrar y analizar datos. Este software hace uso de la *API* (*Application Programming Interface*), la cuál abstrae comandos de bajo nivel y se hace programación a nivel registro[13].

2.2 Sistemas Embebidos

Los sistemas embebidos están presente en el día a día. Ejemplos incluyen máquinas expendedoras, electrodomésticos, teléfonos/teléfonos inteligentes, líneas de fabricación/montaje, televisores, consolas de juegos, automóviles (e.g., dirección asistida y sensores de marcha atrás), conmutadores de red, enrutadores, puntos de acceso inalámbricos, sistemas de sonido, equipos de control médico, impresoras, edificios controles de acceso, parquímetros, medidores inteligentes de energía/agua, relojes, herramientas de construcción, cámaras digitales, monitores, tabletas, lectores electrónicos, cualquier dispositivo robótico, sistemas de acceso/pago con tarjeta inteligente, y más[15].

Los sistemas embebidos pueden verse como un tipo de sistema computacional con *software* integrado, que son diseñados para una aplicación específica[15]. Estos sistemas pueden tener las siguientes características:

- Tienden a estar dedicados a aplicaciones específicas.
- Suelen tener recursos, como poder de procesamiento y memoria, limitados.
- Generalmente son parte de un sistema más grande que pueden estar vinculados a sensores o actuadores externos.
- Suelen tener un rol donde la confiabilidad es crítica, por ejemplo, controles para carros, aviones, y equipo médico.
- Suelen trabajar en tiempo real, donde sus salidas están directamente relacionadas con las entradas, por ejemplo, sistemas de control.
- En los últimos tiempos, la conectividad se ha convertido en una característica principal de los sistemas embebidos, permitiendo a estos ser componentes base del internet de las cosas.

2.3 Linux embebido

El termino de "Linux embebido" hace referencia a una distribución de Linux cualquiera utilizada en un sistema embebido, no hay una distribución específica de Linux embebido[15]. El concepto sistema embebido puede ser explicado de forma simplificada como un sistema de computo con software integrado que fue diseñado para una aplicación especifica[15].

El uso de Linux tiene varias ventajas tanto económicas como técnicas, por lo cuál existe una gran adopción de Linux en sistemas embebidos. Algunas de las razones del crecimiento de Linux embebido han sido su alto rendimiento y estabilidad, su enorme variedad de aplicaciones y protocolos de redes soportados, su escalabilidad, el no tener costo alguno por derechos de autor, el rápido soporte de nuevas arquitecturas de hardware, además de que cada vez más distribuidores de hardware y software son compatibles con Linux[16].

2.3.1 Sistema operativo Debian

Debian es un sistema operativo libre creado por una asociación de programadores voluntarios con el único objetivo de crear *software* libre. Actualmente utiliza el kernel de Linux, creada en un principio por Linus Torvalds y desarrollada por miles de programadores alrededor del mundo. El kernel Linux se encarga de realizar todas las tareas básicas del sistema, permitiendo la ejecución de otros programas, como el explorador de archivos, la terminal y el procesador de textos, por citar algunos ejemplos[17].

Mientras el kernel Linux conforma el núcleo que interactua directamente con el *hardware*, GNU proporciona las herramientas básicas de este sistema operativo, como las ya mencionadas anteriormente. De la combinación de estos dos *softwares* nace el termino GNU/Linux, que prácticamente es a lo que las personas se refieren al decir "distribución de Linux"[17][18].

Debian incluye más de 59 mil paquetes (software precompilado y empaquetado), el gestor de paquetes APT (*Advanced Packaging Tool*), y otras utilidades que hacen posible gestionar fácilmente miles de paquetes en miles de dispositivos (ordenadores o sistemas embebidos)[17].

Así pues, Debian se encarga del correcto funcionamiento de un sistema, gestionando y organizando las tareas, utilizando el núcleo y las demás herramientas básicas para que un usuario pueda ejecutar aplicaciones más complejas en un dispositivo[17].

2.3.2 Beaglebone Black

La Beaglebone Black, mostrada en la Figura 2.2, es una tarjeta que forma parte de la familia *Beagle boards*. Las *Beagle boards* son plaformas de desarrollo *open-source* que utilizan sistemas operativos basados en Linux. Son compactas, de bajo costo, y pueden ser utilizadas para construir aplicaciones complejas donde interaccionen el *software* de alto nivel con los circuitos electrónicos de bajo nivel. Estas tarjetas son plataformas ideales para hacer prototipos de proyectos y diseño de productos que toman ventaja de la libertad y el poder de Linux, combinado con el acceso directo a pines y buses de entrada/salida, permitiendo la comunicación con componentes electrónicos, módulos y dispositivos[15].

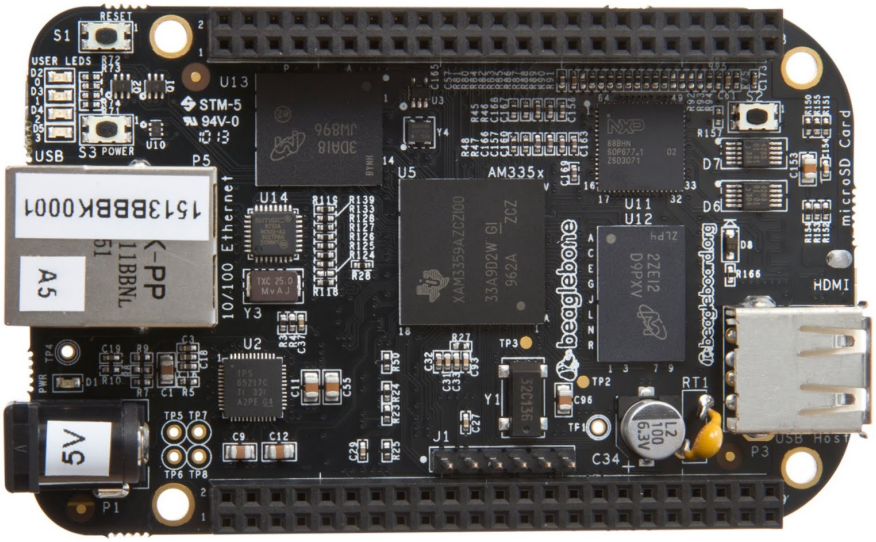


Figura 2.2: Beaglebone Black Rev. C.

La tarjeta *Beaglebone Black* utiliza el microprocesador ARM (Máquina RISC Avanzada, por sus siglas en inglés) Sitara AM335x Cortex-A8 de *Texas Instruments* con una velocidad de 1 GHz. También integra dos PRU (unidad de tiempo real programable, por sus siglas en inglés) de 32 bits, lo que le permite realizar tareas en tiempo real utilizando dos microcontroladores independientes, y utiliza el motor gráfico PowerVR SGX530, capaz de renderizar hasta 20 millones de polígonos en un segundo[15]. Cabe mencionar que el consumo de este procesador cuando está inactivo es de aproximadamente 1 watt, mientras que con cargas pesadas puede llegar a los 2.3 watts.

En cuanto a capacidades de memoria, esta plataforma cuenta con 512 MB de RAM DDR3 para poder realizar las tareas del sistema operativo y el usuario, tiene una eMMC (tarjeta multimedia embebida, por sus siglas en inglés) de 4 GB, para almacenar y ejecutar el sistema operativo, y finalmente una ranura para tarjetas micro SD, con la cuál se puede expandir la memoria de almacenamiento o correr una imagen alterna del sistema operativo desde ésta.

La tarjeta *Beaglebone Black* viene con una imagen del sistema Debian instalada en la eMMC de

fábrica, pero el sistema operativo se puede actualizar o cambiar a otro sistema Linux, como Ubuntu, Android o Angstrom.

En cuestión de periféricos, esta tarjeta con una variedad variedad, comenzando con sus 65 puertos de entrada/salida de propósito general, con los cuales puede generar y leer señales digitales, ocho salidas con modulación de ancho de pulso (PWM), siete entradas analógicas, para poder leer señales analógicas a través del convertidor analógico-digital y 4 temporizadores para hacer tareas en tiempos específicos. Para la comunicación serial cuenta con: 2 puertos I^2C, 4 UART, 2 CAN y 2 SPI.

En cuanto a puertos, la tarjeta *Beaglebone Black* dispone de un puerto USB, para conectarle algún dispositivo o periférico externo, un puerto micro USB de alimentación y datos, para conectarlo a una computadora, un puerto micro HDMI, para conectarle un monitor externo, un puerto Ethernet, para conectarse a una red y una entrada de 5V, para alimentarlo con una fuente externa.

Además de estas características, esta plataforma también puede hacer uso de varias herramientas de utilidad las cuales se mencionan a continuación.

2.3.3 Protocolo Secure Shell

Secure Shell (SSH) es un protocolo de comunicación encriptada y segura entre dos dispositivos[15]. Este método permite iniciar sesión remotamente de una computadora a otra de forma segura. Todas las autenticaciones de usuario, comandos, salidas, y transferencias de archivos son encriptadas para proteger de ataques en la red[19].

El protocolo usa el modelo cliente-servidor, por lo cual la conexión es establecida por el cliente SSH conectándose al servidor SSH. El cliente dirige el proceso de la configuración de la conexión y utiliza criptografía de clave pública para verificar la identidad del servidor SSH. Después de la configuración, el protocolo SSH usa un cifrado simétrico fuerte y algoritmos de hashing para asegurar la privacidad e integridad de la información intercambiada entre el cliente y el servidor[19].

La forma de utilizar SSH para ingresar con un usuario específico a otra computadora remotamente desde una terminal Linux es con el comando "`ssh usuario@servidor`" , a continuación se deberá ingresar la contraseña del mismo usuario.

El protocolo SSH también cuenta con un programa para intercambiar archivos entre cliente y servidor, de forma remota y segura, llamado `scp` (*secure copy*). La forma básica de utilizarlo para copiar un archivo al servidor es: "`scp archivo.ext usuario@servidor:ruta/al/archivo`". Para copiar un archivo desde el servidor se utiliza de la forma: "`scp usuario@servidor:/ruta/al/archivo.ext ruta/local`"[19].

2.3.4 Entorno de desarrollo *Cloud9*

Cloud9, mostrado en la Figura 2.3, es un entorno de desarrollo integrado (*Integrated Development Environment*) basado en la nube que soporta varios lenguajes de programación. Este IDE integra el desarrollo de código con un entorno de ejecución, permitiendo escribir, correr y depurar código local o remotamente[15].

Este IDE viene preinstalado y habilitado en las imagenes de sistema operativo disponibles en la página web `https://beagleboard.org/`, tiene una carga de procesamiento suficientemente baja para

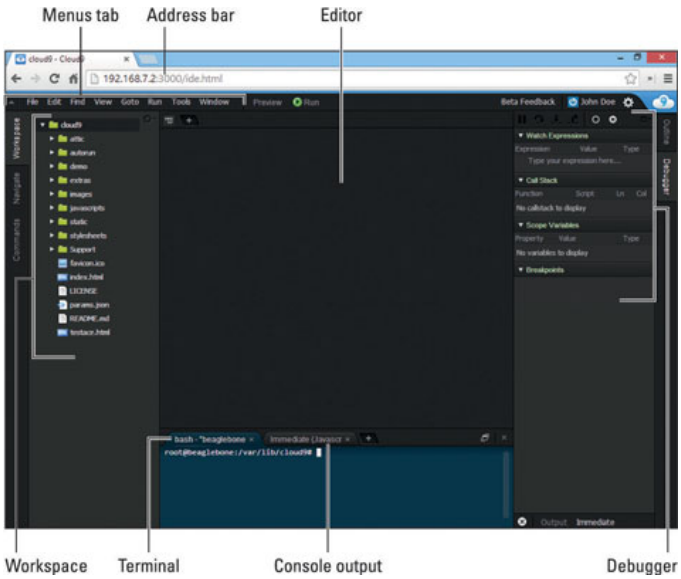


Figura 2.3: Entorno de desarrollo integrado *Cloud9* [20].

poder correr desde una *Beagle board* y puede ser accedido remotamente desde una computadora a través de un navegador web, por lo que no es necesario instalar alguna aplicación adicional. Esto a su vez elimina el uso restrictivo dentro de algún sistema operativo específico. Para poder utilizar este IDE de forma remota es necesario introducir la IP de la *Beaglebone* en la barra de dirección del navegador web. Opcionalmente, también se puede conectar a la *Beaglebone* con un cable *Ethernet* e introducir el *hostname* de ésta[20].

Cabe aclarar que aunque la interfaz gráfica se muestra a través de en un ordenador local, el IDE se ejecuta desde la *Beagle board* a través de un servidor web local, alojando en ésta los proyectos con sus respectivos archivos.

En definitiva, *Cloud9* es una herramienta de gran utilidad para llevar a cabo el desarrollo de nuestros proyectos de programación utilizando plataformas de Linux emebido como lo son las *Beagle boards*

2.4 Librerías útiles del lenguaje C

Dentro del entrono de desarrollo en lenguaje C hay dos librerías que son de gran utilidad para poder realizar tanto comunicación serial (`termios.h`), como para tener una noción del tiempo, hora y fecha (`time.h`), ambas se explican a mayor profundidad en seguida en su sección correspondiente.

2.4.1 Librería `termios.h`

La librería `termios.h` es una interfaz general de la terminal para controlar puertos de comunicación asíncrona para el lenguaje de programación C. Esta librería poseé una estructura con los parámetros de la comunicación, los cuales pueden ser modificados, así como funciones de transmisión y recepción de datos y control de la comunicación.

La estructura `struct termios`, definida dentro de la librería `termios.h`, contiene los parámetros de al menos el manejo de los datos de entrada y salida, los modos de control de la comunicación, los modos de funcionamiento local y los caractéres especiales que pueden ser utilizados como comandos de control de la comunicación[21].

Los parámetros de la comunicación dentro de la estructura `struct termios`, que son manejados como un arreglo de banderas conformando una mascara de bits, pueden ser obtenidos y establecidos por medio de las funciones `tcgetattr()` y `tcsetattr()` respectivamente, dichas funciones están incluidas en la librería `termios.h`[21]. De esta forma, una vez habiendo configurado el puerto de comunicación estableciendo los parámetros, para transmitir y recibir datos es cuestión de escribir y leer el puerto a través de su archivo asociado en el sistema de archivos de Linux.

2.4.2 Librería `time.h`

La librería `time.h`, un archivo de cabecera de la biblioteca estándar del lenguaje de programación C, proporciona funciones para manipular y obtener información sobre fechas y horas. Incluye macros, que son constantes simbólicas definidas por el preprocesador, y una variedad de funciones y variables que permiten realizar mediciones de intervalos de tiempo, así como gestionar información de la hora, fecha e incluso horario de verano de la hora local, de otras regiones o diferentes épocas.

La librería define varios tipos de variables para almacenar diferentes medidas de tiempo. El tipo `time_t` representa el tiempo en segundos desde la época de Unix. Para una mayor precisión, se utiliza la estructura `timespec`, que incluye los miembros `tv_sec` (segundos) y `tv_nsec` (nanosegundos). La estructura `tm` se utiliza para representar un tiempo de calendario, con miembros como `tm_year`, `tm_mon`, `tm_mday`, `tm_hour`, `tm_min`, `tm_sec`, entre otros, que almacenan los componentes de una fecha de forma desglosada[22].

`time.h` incluye diversas funciones para manipular fechas y horas:

Obtener el tiempo:

- `clock()`: Determina el tiempo de utilización del procesador, expresado en ciclos de reloj. Devuelve un valor de tipo `clock_t`.
- `time()`: Determina el tiempo codificado de calendario actual, expresado en segundos desde la época de Unix.

Convertir el tiempo:

- `asctime()`: Convierte una estructura tm en una cadena de caractéres que representa la fecha y hora.
- `ctime()`: Convierte un tiempo codificado (`time_t`) en una cadena de caractéres.
- `gmtime()`: Convierte un tiempo codificado a una estructura tm expresado en UTC.
- `localtime()`: Convierte un tiempo codificado a una estructura tm expresado en la hora local.
- `strftime()`: Formatea una fecha y hora en una cadena de caractéres según una cadena de formato especificada.

Calcular diferencias de tiempo:

- `difftime()`: Calcula la diferencia entre dos tiempos codificados.

Otras funciones:

- `mktime()`: Convierte una estructura tm en un tiempo codificado.
- `timespec_get()`: Obtiene la hora actual con precisión de nanosegundos.

En resumen, la librería time.h proporciona un conjunto completo de herramientas para trabajar con fechas y horas en C, permitiendo a los desarrolladores obtener, manipular, formatear y convertir información temporal de diversas maneras.

2.5 Transmisor-Receptor Asíncrono Universal

El UART (Universal Asynchronous Receiver-Transmitter, por sus siglas en inglés) es un protocolo de comunicación serial creado para transmitir y recibir datos entre dos dispositivos. Esta comunicación es de forma bidireccional, de modo que se utilizan dos hilos (Tx y Rx). En la Figura 2.4 se muestra un ejemplo de la conexión entre dos dispositivos. La comunicación también es asíncrona que, a diferencia de la comunicación sincrónica, no requiere transmitir una señal de reloj ya que cada dispositivo cuenta con su propio reloj. Por lo general los relojes utilizados en este dispositivo tienen una frecuencia múltiplo de la velocidad transmisión, para así poder tomar muestras en el medio de los bits que se reciben.

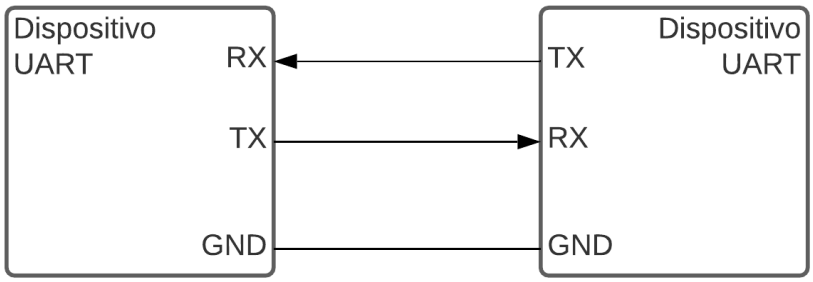


Figura 2.4: Conexión de dos dispositivos UART.

El dispositivo UART convierte el dato a transmitir en una serie de bits, haciendo uso de un registro de desplazamiento, el cuál es implementado con varios flip-flops conectados en cascada. De la misma forma, el receptor va desplazando los bits recibidos para reconstruir el dato original.

Para realiza una transmisión de un dato, el protocolo de comunicación establece que ésta debe comenzar con un bit de inicio, el cual es captado por el receptor, indicándole a este el inicio de una transmisión. De esta forma el receptor almacenará los siguientes bits recibidos. Los bits del dato a transmitir suceden al bit de inicio uno por uno, y una vez transmitidos éstos, la comunicación termina

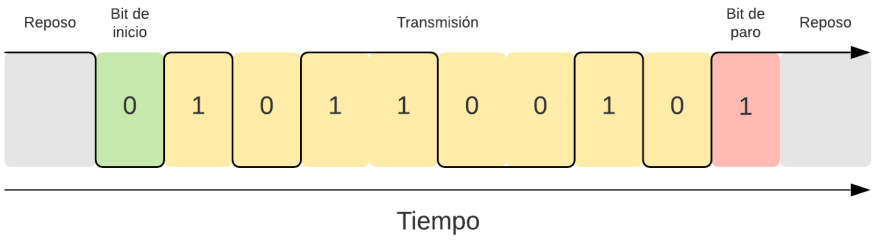


Figura 2.5: Proceso de transmisión a un dispositivo UART.

con el bit de paro para en seguida pasar de nuevo al estado de reposo. En la Figura 2.5 se muestra de una manera gráfica como se lleva a cabo una transmisión de un dato a un dispositivo UART.

Todos los bits transmitidos en esta comunicación tienen una duración determinada por la velocidad de transmisión. Los bits de inicio y paro de cada transmisión son generados por el dispositivo transmisor y removidos por el receptor. La transmisión puede ser comenzada en cualquier momento, siempre y cuando no se esté realizando ya una.

La comunicación entre dos dispositivos UART puede ser configurada con una gran cantidad de opciones, pudiéndose ajustar a las necesidades que se requieran para la comunicación. Se debe de tener el cuidado de que la configuración entre ambos dispositivos sea compatible para evitar errores en la comunicación.

La velocidad de transmisión es uno de estos parámetros configurables de la comunicación. Por lo regular estas velocidades pueden ajustarse comenzando desde 300 baudios (bits por segundo) con algunos valores intermedios hasta llegar a 115,200 baudios. Generalmente se utiliza una velocidad de 9,600 baudios. También es posible configurar parámetros de detección de errores en la transmisión, como el bit de paridad, el cuál indica si el la cantidad de “1”s del dato es par. Otro de los parámetros que pueden ser ajustados es la longitud de los datos, siendo ocho el número máximo de bits, si se desea utilizar dos bits de paro, entre otras cosas.

2.6 Sensores de concentración de gas

Los sensores de concentración de gases son sensores electroquímicos, donde un gas específico reacciona con un electrólito. Esta reacción electroquímica a su vez produce una corriente eléctrica, que al ser dependiente de la concentración de gas puede ser medida con un dispositivo electrónico para estimar la concentración de dicho gas[23]. También existen sensores de gases de flujo UV, que al no utilizar algún químico que reaccione al gas pueden tener una vida útil mayor y evitan el uso de sustancias peligrosas.

2.6.1 Controladores de sensores

Los controladores de sensores son dispositivos eléctricos que se encargan del suministro de energía necesaria al sensor de gas, de la medición e interpretación corriente eléctrica causada por la reacción química del sensor, en algunas ocasiones del acondicionamiento de la señal, y de enviar estas mediciones a algún otro dispositivo. Estos controladores son enormemente utilizados en los módulos de sensores

ya que además de simplificar en la interfaz con los sensores, pueden reducir el ruido al enviar las mediciones.

2.6.2 Sensor de oxígeno *LuminOx Optical Oxygen LOX-02*

El sensor *LuminOx Optical Oxygen*, mostrado en la Figura 2.6 es un sensor óptico de oxígeno de bajo costo, bajo consumo y de larga vida útil, desarrollado y fabricado por la compañía *SST*. Además de medir oxígeno, este sensor es también capaz de realizar mediciones de la temperatura y la presión barométrica del gas, siendo estos parámetros útiles para tener un mayor conocimiento del gas, o la atmósfera donde se encuentra éste. Este sensor puede captar una concentración de oxígeno de hasta 25 %, y 300 mbar de presión parcial del oxígeno, con un error menor al 2 %. Puede funcionar en un ambiente con un rango de temperaturas dentro de -30 a 60 grados Celsius, humedad relativa de hasta el 99 %, siempre y cuando no haya condensación, y desde 500 mBar hasta 1200 mBar de presión.

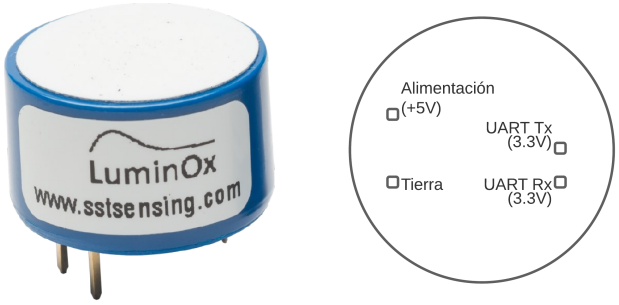


Figura 2.6: Sensor *LuminOx Optical Oxygen*[24] y su esquema de pines visto desde su parte inferior.

Este sensor tiene un controlador con una interfaz que le permite conectarse a un microcontrolador a través de un UART con los parámetros preestablecidos de 9600 baudios de velocidad, datos con tamaño de 8 bits, un bit de paro y sin hacer uso del bit de paridad. El Controlador debe ser alimentado con 5 volts (teniendo $\pm$ 0.5 volts de tolerancia) y el consumo de corriente puede tener picos de 20 mili amperios. La salida de la señal de comunicación UART es TTL con un nivel alto de 3.3 volts. Estas conexiones y la posición de los pines son mostradas en la Figura 2.6.

El controlador de este sensor se maneja por medio de comandos, con los cuales se puede elegir un modo de funcionamiento con el comando "`M`" seguido de un espacio y el argumento "`1`", "`2`", o "`0`". Los modos de funcionamiento son elegibles por medio de los comandos mencionados anteriormente, y estos se pueden establecer dependiendo de la aplicación que se le dé a este sensor, o se puede cambiar a otro en base a las circunstancias. Estos modos se describen a continuación:

- *Stream*: El sensor envía todas las mediciones y el estado de este aproximadamente una vez cada segundo. Es activado por defecto al encender el sensor y se selecciona con el comando "`M 0`".
- *Poll*: El sensor continua tomando mediciones, pero sólo las envía al ser solicitadas. Este modo se selecciona con el comando "`M 1`".
- *Off*: El sensor deja de tomar mediciones y logra reducir el consumo de corriente hasta 6 mili amperios. Este modo se selecciona con el comando "`M 1`".

Para solicitar mediciones en el modo *Poll* (`M 1`), se utiliza el comando "`%`" para el porcentaje de oxígeno, y "`O`" para obtener la presión parcial de oxígeno (ppO_2) en milibars. Las demás mediciones

se solicitan con los comandos "`T`", para obtener la temperatura (en grados Celsius), "`P`", para la presión barométrica (en mBar), y para obtener todas las mediciones se utiliza el comando "`A`". A su vez, también hay comandos para ver el estado (comando "`e`") y la información ("`#`") del sensor. Los comandos enviados a este sensor deben tener la terminación "`\r\n`" (retorno de carro y salto de linea) para ser identificados debidamente.

2.6.3 Sensor de monóxido de carbono CO-AF

El sensor *CO-AF*, mostrado en la Figura 2.7, es un sensor electroquímico que utiliza tecnología de pila de combustible (fuel cell) y de electrodos, probada con estabilidad a largo plazo y operación confiable diseñado por la compañía *Alphasense*. El sensor usa el tamaño estándar de 20 milímetros de diámetro y su fabricante garantiza una correcta estimación de las mediciones hasta las 5,000 ppm de monóxido de carbono, operando también en los rangos de desde -30 hasta los 50 grados Celsius de temperatura, de 80 a 120 kilopascales de presión y una humedad relativa del 15 % al 90 %.

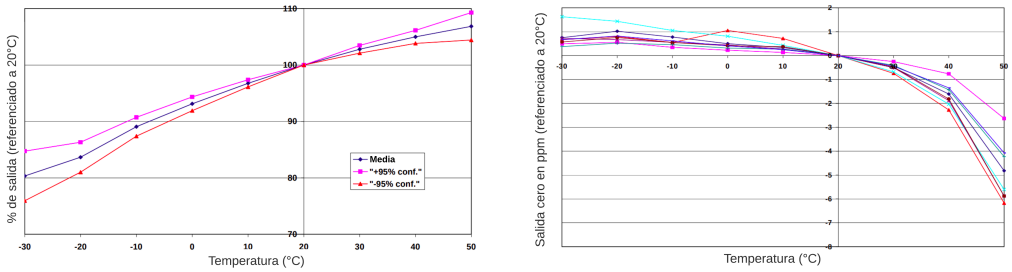
Figura 2.7: Sensor de monóxido de carbono CO-AF de *Alphasense*[25].

Este sensor utiliza una salida de corriente proporcional a la cantidad de monóxido de carbono. La sensibilidad de éste está en un rango de 55 a 90 nanoamperios por cada 400 ppm del gas, también tiene un tiempo de respuesta menor a 25 segundos midiendo un escalón de cero a esta misma cantidad de *CO*. La sensibilidad de este sensor tiene una dependencia a la temperatura, esta puede ser compensada con un controlador para evitar así la susceptibilidad a la temperatura de las mediciones. La forma en la que esta sensibilidad es afectada por la temperatura se muestra con mayor detalle en la Figura 2.8.

En cuanto a su vida operativa, el fabricante *Alphasense* asegura que la señal de salida del sensor se reducirá un máximo de 20 % con respecto a la respuesta original en un periodo mayor de 24 meses, y puede almacenarse hasta por seis meses en un recipiente sellado dentro de un rango de temperaturas de 3 a 20 grados Celsius.

2.6.4 Tarjeta controladora EC200

La tarjeta EC200, mostrado en la Figura 2.9, es un controlador de alto rendimiento y bajo consumo compatible con una amplia variedad de sensores de gas eletroquímicos montables de 20 milímetros *(20mm electro-chemical plug-in gas sensor cells)*. Esta tarjeta controladora proporciona la interfaz necesaria con el sensor de gas, convierte la salida de corriente del sensor en voltaje, el cual es medido con un convertidor analógico-digital de alta resolución. A este voltaje digital, se le corrige y compensa la presión y temperatura con un microcontrolador para producir una concentración precisa de gas medida en partes por millón[26].



(a) Variación en la sensibilidad causada por cambios de la temperatura. Se muestran media y los intervalos de +- ±95 % de confianza.

(b) Variación de la salida cero causada por cambios en la temperatura expresado en ppm, referenciado a cero en 20 °C

Figura 2.8: Gráficas de la dependencia a temperatura de la sensibilidad de un lote típico de sensores CO-AF[25].

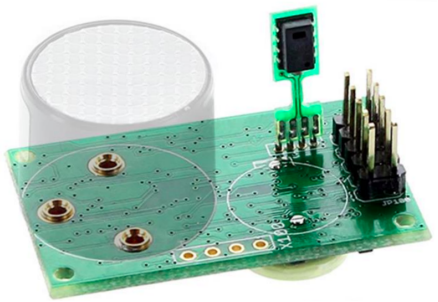


Figura 2.9: Tarjeta controladora de sensores electroquímicos EC200 [26].

La tarjeta controladora también integra sensores de temperatura/humedad y de presión barométrica, con los cuáles obtiene mediciones de estas magnitudes y las utiliza para corregir y compensar la estimación de concentración de gas. Ésta además realiza mediciones del el voltaje obtenido de la corriente generada por sensor, el cuál es filtrado con un filtro pasa-bajas para remover perturbaciones transitorias. Tanto las mediciones del voltaje como de la concentración de gases compensada son obtenidas con y sin filtrado, esta última también es posible generarla sin filtrado ni compensación. Todas estas mediciones son guardadas en la memoria del controlador y pueden ser transmitidas a través de su interfaz UART.

La tarjeta EC200 puede operar correctamente en los rango de -30 hasta 60 °C de temperatura, desde 50 hasta 115 kilopascales de presión, y hasta 99 % de humedad relativa sin haber condensación. Este puede ser alimentado con un voltaje de entre 3.2 y 5 volts, consumiendo una corriente que puede llegar hasta los 10 miliamperios de pico. En la Figura 2.10 se indican los pines de alimentación del sensor, así como las terminales de la interfaz UART.

La interfaz UART del controlador EC200 opera a una velocidad de 9600 baudios, enviando datos con tamaño de ocho bits sin implementar bit de paridad, esta interfaz utiliza tanto terminales Tx y Rx, con un nivel lógico CMOS de 3.3V (la terminal Rx soporta hasta 5V), así como RS485 diferencial. Cada línea de datos recibida y transmitida por el controlador termina con un salto de linea y retorno de carro (“`\r\n"`). El controlador, además de tener una interfaz de comunicación serial, puede utiliza salidas de PWM y analógica, las cuales tienen que ser configuradas para poder utilizarse.

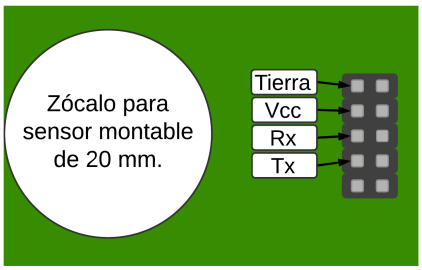


Figura 2.10: Esquema de pines de la tarjeta controladora EC200.

Los modos de operación del controlador son: *polled* y *streaming*. En el modo *polled*, el sensor sólo transmite datos cuando responde a un comando. En el modo *streaming*, el sensor transmite mediciones en intervalos de un segundo, este intervalo puede ser modificado. Estos dos modos de operación pueden ser configurados con el comando "K".

Los comandos son enviados al controlador por medio de la interfaz UART. Estos pueden ser utilizados para extraer las mediciones obtenidas por éste, obtener información a cerca del sensor o de él mismo, o realizar configuraciones y ajustar sus parámetros de operación. En caso de haber un error en la recepción de algún comando, el controlador responde con la letra 'E' seguido del código de error.

Los comandos para obtener mediciones son los siguientes:

- B - presión barométrica (compensada con la temperatura) milibars.
- b - presión barométrica (ADC).
- H - humedad relativa.
- J - voltaje entrada analógica.
- j - voltaje bruto.
- Q - leer mediciones.
- T - temperatura.
- t - temperatura bruta de barómetro.
- V - voltaje con filtrado.
- v - voltaje sin filtrado.
- z - concentración de gas sin filtrado.
- Z - concentración de gas con filtrado.

Los comandos para realizar configuraciones son los siguientes:

- C - configura la hora y fecha.
- K - configura modo de comunicación.

- M - campos de salida (seleccionar mediciones).
- P - set valor de un parámetro.
- r - borrar memoria de registro.
- U - calibración de cero.
- u - calibración manual de cero.

Los comandos para obtener información son los siguientes:

- c - leer hora y fecha.
- G - tipo de sensor.
- p - leer parámetro.
- R - leer memoria de registro.
- w - reiniciar parámetros a valores de fábrica.
- X - calibrar a una concentración específica.
- Y - reportar identificador del dispositivo.
- . - obtener multiplicador de la medición.

2.6.5 Sensor de bióxido de carbono *SprintIR-W*

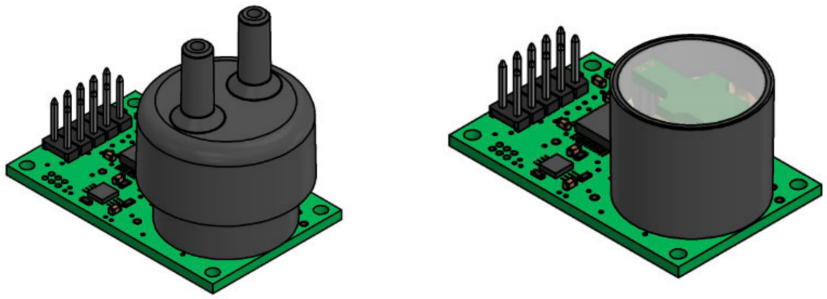
El sensor *SprintIR* de la compañía *Gas Sensing Solutions*, mostrado en la Figura 2.11, mide concentraciones de CO_2 utilizando tecnología óptica LED NDIR (infrarrojo no dispersivo, por sus siglas en inglés), lo que lo hace un sensor confiable y duradero. Este sensor está diseñado para desempeñarse en tareas de monitoreo y análisis de CO_2 en tiempo real. A menudo es utilizado en áreas como el cuidado de la salud, empaquetado de alimentos, ciencia del deporte y sistemas contra incendios[27].

Figura 2.11: Sensor de bióxido de carbono SprintIR®-W [28].

Este sensor es capaz de realizar 20 lecturas por segundo, teniendo una alta frecuencia de muestreo y una gran velocidad de respuesta (las mediciones validas son obtenidas 1.2 segundos después de haber encendido el sensor). El rango de medición de este sensor puede llega hasta el 100 % de concentración de CO_2 en algunas variantes. También existen otras opciones con los rangos de medición: 0 a 5 %, 0

a 20 %, 0 a 60 %. Las mediciones se realizan con una precisión típica de 70 ppm para el rango de 0 a ±60 %, y ±300 ppm para 0 a 100 %[27].

El sensor *SprintIR* puede permitir un gran paso de flujo, pudiendo utilizarse con un adaptador de flujo, como se muestra en la Figura 2.12(a), o simplemente utilizar el sensor con la cubierta de membrana como se muestra en la Figura 2.12(b). El adaptador de flujo puede ser personalizado eligiendo distintas opciones ofrecidas por el fabricante.



(a) Sensor SprintIR con adaptador de flujo.

(b) Sensor SprintIR sin adaptador de flujo.

Figura 2.12: Sensor SprintIR con y sin adaptador de flujo[27].

Este sensor incorpora un sistema, el cuál permite el control y transmisión de datos por medio de UART, así como puesta en cero automática y un acondicionamiento de las mediciones. El diagrama de bloques de este sistema y sus componentes puede visualizarse en la Figura 2.13.

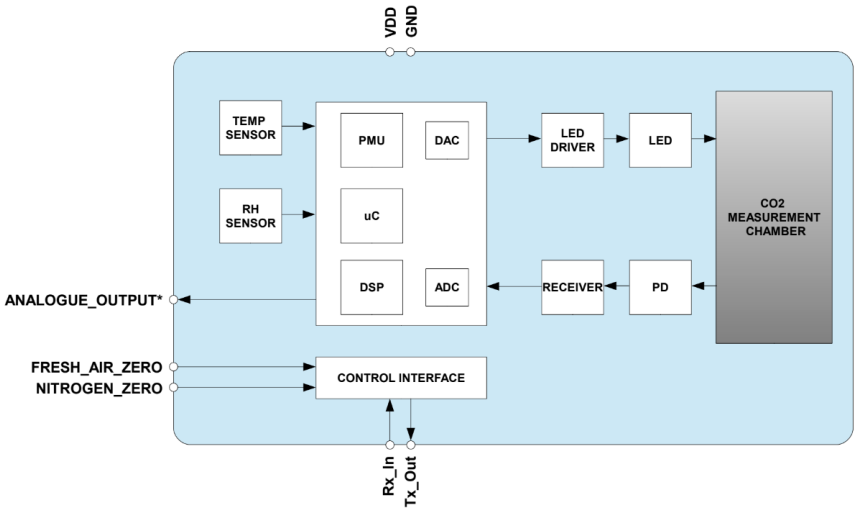


Figura 2.13: Diagrama a bloques del sensor SprintIR®-W [27].

Los sensores de temperatura y humedad mostrados en la Figura 2.13, así como la salida de señal analógica, son opcionales, por lo que no todas las variantes del sensor *SprintIR* los integran.

El acondicionamiento de las mediciones se realiza con un procesador de señales digitales (DSP, por sus siglas en inglés, *Digital Signal Processor*). Su comportamiento puede ser modificado dependiendo de la velocidad de respuesta necesaria, o la cantidad de ruido que se requiera filtrar.

El control de este sensor se realiza por medio de comandos que son recibidos por éste utilizando el puerto UART. Los comandos utilizan la codificación ASCII y deben terminar con un retorno de carro y un salto de linea("`\r\n`"). El sensor responderá con un signo de interrogación (?) si el comando recibido no es válido o éste no se ha recibido con algún error.

El sensor *SprintIR* tiene tres modos de operación que pueden ser seleccionados por medio del comando "`K #\r\n`" (donde `#` es el número de modo seleccionado). Los modos de operación del sensor de explican a continuación.

- *SLEEP*: En este modo el sensor está en espera de comandos. El sensor no realiza ninguna medición, por lo tanto al recibir un comando este puede responder rápidamente. Este modo se establece con el comando "`K 0\r\n`".
- *STREAM*: En este modo las mediciones son reportadas dos veces por segundo. Los comandos son procesados cuando se reciben, pero si se encuentra realizando una medición puede haber un retraso de 10 ms en la respuesta. El sensor inicia por defecto en este modo al ser encendido y puede establecerse con el comando "`K 1\r\n`".
- *POLLING*: En este modo el sensor reporta mediciones sólo cuando son solicitadas, pero las continua realizando en segundo plano. Este modo se puede establecer con el comando "`K 2\r\n`".

La interfaz UART de este sensor opera a una velocidad de 9600 baudios, enviando datos de 8 bits con un bit de paro y sin utilizar bit de paridad, ni control de flujo por *hardware*.

El sensor se alimenta con 3.3V, soportando hasta 5.5V y pudiendo consumir hasta 40 mA (de pico) durante el encendido. El consumo de corriente puede llegar a los 35 mA cuando se están tomando muestras. Si el sensor es configurado en modo *SLEEP*, el consumo de corriente puede disminuir hasta los 0.01 mA. Las terminales del puerto UART, así como la alimentación del sensor, se muestran en la Figura 2.14. Las terminales Tx y Rx operan con un nivel lógico de 3.3 volts.

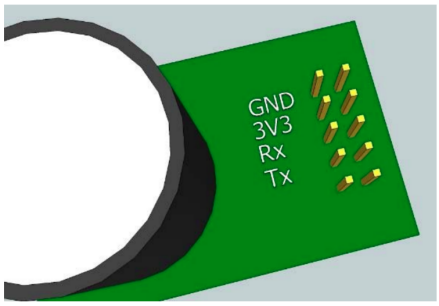
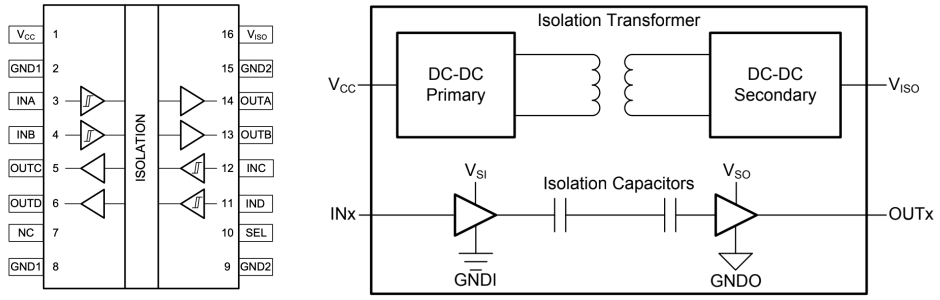


Figura 2.14: Salidas de la interfaz UART del sensor de CO_2 *SprintIR*[28].

Este sensor puede operar en los rangos de temperatura de 0 a 50 °C, de 0 a 95 % de humedad y desde 500 mbar hasta 2 bar de presión. El fabricante no recomienda utilizar el sensor fuera de los rangos de operación, ya que al someterlo a estrés puede sufrir daños irreversibles.

2.7 Aislador digital ISOW7842

El circuito integrado ISOW7842 es un aislador digital que forma parte de la familia ISOW784x del fabricante *Texas Instruments*. Cuenta con cuatro canales (dos en cada dirección), como se muestra en la Figura 2.15(a), y un convertidor DC-DC, mostrado en el esquema simplificado de la Figura 2.15(b). Este convertidor proveé alimentación aislada, la cuál se puede utilizar para energizar un circuito eliminando la necesidad de una fuente de alimentación separada, lo cuál es útil para diseños con espacio reducido. Este CI puede ser utilizado en las áreas: automatización industrial, control de motores, redes eléctricas, equipos médicos, y pruebas y mediciones.



(a) Canales del CI ISOW7842

(b) Esquema simplificado de los CI ISOW784x.

Figura 2.15: Esquemas del CI ISWO7842[29].

Las características principales de este CI se enlistan a continuación:

- Velocidades de transmisión de hasta 100 Mb por segundo.
- Robusta barrera de aislamiento con una vida estimada mayor a 100 años con un voltaje de operación de $1kV_{RMS}$, pudiendo aislar hasta $5000kV_{RMS}$ y una capacidad de sobre tensión de $10kV_PK$.
- Convertidor CD-CD de alta eficiencia integrado con transformador de aislamiento en chip.
- Amplio rango de voltaje de alimentación, desde 3V hasta 5.5V.
- Potencia de salida máxima de 0.65 W, con una corriente disponible de 130 mA a 5 V y 75 mA a 3.3 V.
- Protección a sobrecarga y corto circuito, apagado por calentamiento, y arranque suave para evitar entradas excesivas de corriente.
- Rango de temperatura de operación extendido, de $-40\,°C$ a $125\,°C$.
- Compatibilidad electromagnética robusta con alta inmunidad y bajas emisiones.

2.8 Conclusión

Este capítulo ha establecido el marco teórico necesario para el desarrollo de un sistema embebido basado en Linux para la medición de concentraciones de gases en reacciones sólido-gas. Se han revisado

conceptos clave de sistemas de adquisición de datos, sistemas embebidos, Linux embebido, comunicación UART, sensores de gases y circuitos de protección, justificando su relevancia para el sistema propuesto. Este marco teórico proporciona la base para las decisiones de diseño y la implementación práctica que se detallarán a continuación. Si bien la integración de estos elementos presenta desafíos específicos, el conocimiento adquirido sienta las bases para un desarrollo exitoso. El siguiente capítulo abordará el diseño e implementación del sistema, presentando la arquitectura, los componentes seleccionados, las pruebas de verificación se presentan en el Capítulo 4.

Capítulo 3

Desarrollo del sistema

En este capítulo se presenta una descripción detallada del sistema de adquisición de datos desarrollado en este trabajo y sus elementos, tanto de *hardware* como de *software*, así como su funcionamiento, y su utilización. Dentro de los temas que se abordan están: descripción de la *single-board computer* utilizada y sus configuraciones, sensores utilizados y su configuración, la atmósfera aislada implementada, el circuito de protección utilizado, descripción del programa para la adquisición de datos y sus funciones utilizadas, librerías desarrolladas, ejemplo del archivo de datos generado, con los cuales se podrá tener un buen entendimiento profundo de este sistema y utilizarlo o replicarlo.

El sistema implementado en este trabajo, mostrado en la Figura 3.1, es capaz de recabar información de una atmósfera o un ambiente, obteniendo las mediciones de las concentraciones de CO, CO_2 y O_2, además de la temperatura, presión y humedad relativa, por medio de sensores controlados por una *single-board computer Beaglebone*.

Este sistema consta de una *single-board computer* (computadora de una sola placa) *Beaglebone Black*, que es accedida por otro dispositivo de forma remota a través del protocolo Secure Shell (SSH). También integra los sensores *CO-AF* (CO), *SprintIR* (CO_2) y *LuminOx* (O_2), y un circuito de aislamiento para conectar los sensores a la *Beaglebone* de forma segura. Dentro de la *Beaglebone* se encuentran los archivos del código fuente del programa para la adquisición de datos, así como el archivo de datos del registro, donde quedan guardadas las mediciones una vez que éstas ha sido realizadas.

El funcionamiento de este sistema puede ser adaptable, pudiendo establecer los parámetros de la duración de la adquisición y su periodo de muestreo al iniciarla desde la terminal de la *Beaglebone*.

Para alimentar al sistema se utiliza una fuente de CD de 5 volts con 2 amperios y se conecta a una red por medio de un cable *Ethernet* (pudiendo conectarse con *Wi-Fi*, en el caso de disponer de una tarjeta *WLAN*) para poder hacer uso del protocolo *SSH*.

Para realizar mediciones de prueba, los sensores fueron montados dentro de una atmósfera aislada (parcialmente). Este sistema se prueba con dos métodos, donde se introducen a la atmósfera aislada los gases de exhalación de una persona en (1), y en (2) se introduce una vela encendida (combustión). Estas pruebas se describen en el capítulo 4, donde también se muestran los resultados de las mismas.

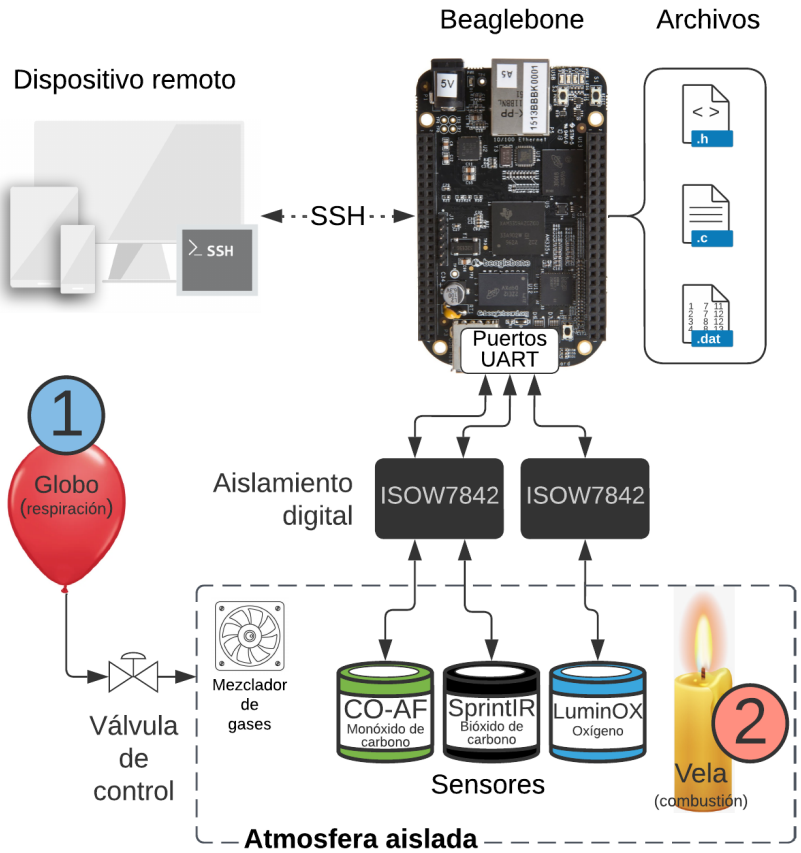


Figura 3.1: Diagrama del sistema desarrollado.

3.1 Beaglebone Black

Como ya se mencionó anteriormente, este sistema utiliza la *single-board computer* ***Beaglebone Black***, mostrada en la Figura 3.2, con la cuál se controlan los sensores y se registran las mediciones obtenidas de éstos. Esta tarjeta cuenta con una generosa variedad de periféricos y un sistema operativo basado en *Linux*, lo que le da gran versatilidad al sistema, pudiendo adaptarlo a las necesidades de la adquisición de datos y siendo posible además agregar más funciones para diferentes adquisiciones de datos.

La *Beaglebone* controla a los sensores de gas a través de comandos, con los cuales los configura y les solicita mediciones; también se puede obtener información de los sensores. Dichos comandos son enviados a través de los puertos *UART*, con los cuales la *Beaglebone* se comunica con los sensores. La gestión de esta comunicación, así como control de los sensores y el registro de las mediciones, se realiza con un programa desarrollado para este sistema, el cuál se ejecuta en la *Beaglebone*. El programa y la librería utilizada para la comunicación a través de UART, así como sus funciones, se explican a detalle en la Subsección 3.5.1.

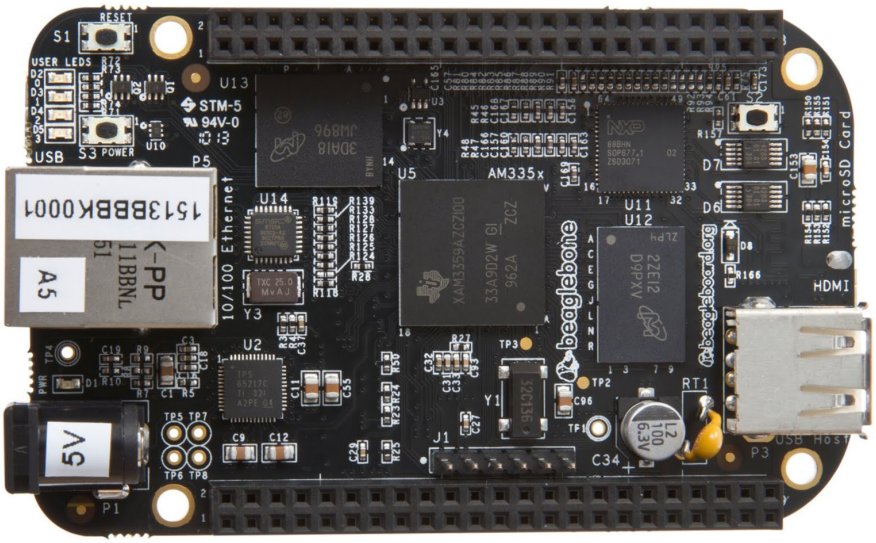


Figura 3.2: Beaglebone Black Rev. C.

En este sistema implementado, la *Beaglebone Black* puede utilizarse con cualquier dispositivo que pueda hacer uso del protocolo SSH. Por medio de éste, se accede a la terminal para ejecutar comandos `bash`. De esta forma es posible correr el programa de adquisición de datos, obtener el archivo de datos con los resultados de la adquisición, o modificar el código fuente del programa (en caso de ser necesario).

En esta aplicación, la *Beaglebone Black* utiliza el sistema operativo *Debian*, específicamente la versión *Debian 9.9 Stretch LXQT* (con interfaz gráfica). Este sistema operativo está instalado en una memoria de 16 GB insertada en la ranura para tarjetas micro SD de la *Beaglebone*. De esta forma el sistema no está tan limitado con el espacio de almacenamiento y se pueden instalar actualizaciones y software adicional en caso de ser necesario. La imagen del sistema operativo puede ser descargada desde la página oficial de *BeagleBoard* (beagleboard.org).

3.1.1 Configuraciones realizadas en la *Beaglebone*

Antes de utilizar la *Beaglebone Black* en esta aplicación, se realizaron algunas configuraciones y actualizaciones para facilitar su utilización, estos cambios se enlistan a continuación:

- Se cambio el *hostname* a beagleboard, con el comando `sudo hostnamectl -static set-hostname beagleboard` para poder diferenciarla en caso de haber otras Beaglebone en la misma red.
- Se configuró la hora local ejecutando el comando `sudo dpkg-reconfigure tzdata` y eligiendo la zona horaria de México para un manejo correcto de la hora y fecha.
- Se expandió la partición del sistema de archivos para poder utilizar todo el espacio disponible de la tarjeta micro SD, actualizando el *script* `grow_partition` y después ejecutándolo como se muestra en el Listado 3.1.

- Se ejecutó el comando `sudo tools/update_kernel.sh` para actualizar los *scripts* de *booteo* y *kernel* de *Linux* y así funcionaran de manera óptima.
- Se actualizaron componentes de la distribución con el comando `sudo apt update` y `sudo apt upgrade` y también `sudo apt install -y ti-tidl mjpg-streamer-opencv-python` para el correcto funcionamiento de estos.
- Se automatizó la configuración de los pines de los puertos UART durante el arranque del sistema. Para esto se tuvo que crear el *script* de `bash config-uart-pins.sh`, mostrado en Código 3.2, en la ruta: `/usr/bin`; y un "servicio del sistema" (`ebb-set-uart-pins.service`), mostrado en el Código 3.3, en la ruta: `/lib/systemd/system`. Este servicio del sistema se probó y se instaló utilizando los comandos del Código 3.4. De esta forma, éste se ejecuta durante el arranque y llama al *script* (`config-uart-pins.sh`).

  Los archivos mencioandos en el último punto se encuentran alojados en **TBD**, al que se puede acceder desde el enlace: TBC:https://gitlab.com/sayeth.rd/daq-serial-bbb/resources.

Código 3.1: Comandos para expandir la partición del sistema de archivos.

```
cd /opt/scripts
git pull
sudo /opt/scripts/tools/grow_partition.sh
```

Código 3.2: Archivo `config-uart-pins.sh`.

```
#!/bin/bash
#uart1
config-pin P9_24 uart
config-pin -q P9_24
config-pin P9_26 uart
config-pin -q P9_26

#uart2
config-pin P9_21 uart
config-pin -q P9_21
config-pin P9_22 uart
config-pin -q P9_22

#uart4
config-pin P9_13 uart
config-pin -q P9_13
config-pin P9_11 uart
config-pin -q P9_11
```

Código 3.3: Archivo `ebb-set-uart-pins.service` de servicio del sistema.

```
[Unit]
Description=Enable the UART pins on boot
After=generic-board-startup.service

[Service]
Type=simple
User=root
WorkingDirectory=/usr/bin
ExecStart=/usr/bin/config-uart-pins.sh

[Install]
WantedBy=multi-user.target
```

Código 3.4: Comandos para probar e instalar el servicio `ebb-set-uart-pins.service` en la *Beaglebone*.

```
cd /lib/systemd/system
sudo su
systemd-analyze verify ebb-set-uart-pins.service
systemctl daemon-reload
systemctl enable ebb-set-uarts-pins.service
```

3.2 Sensores de concentración de gases

En este sistema desarrollado se utilizan tres sensores de concentración de gases diferentes, cada uno con su respectivo circuito de control. Los sensores utilizados en este sistema son: el sensor de monóxido de carbono **CO-AF** de la compañía *Alphasense*, mostrado en la Figura 3.3(a), el sensor de bióxido de carbono **SprintIR** de la compañía *Gas Sensing Solutions*, mostrado en la Figura 3.3(b) y el sensor de oxígeno **LuminOx Optical Oxygen LOX-02** del fabricante *SST*, mostrado en la Figura 3.3(c). Con estos sensores, el sistema es capaz de medir concentraciones de hasta 250,000 partes por millón (25 %) de O_2, 5,000 partes por millón (0.5 %) de CO, y 200,000 partes por millón (20 %) de CO_2. Algunos de estos sensores también pueden medir la temperatura, humedad y presión del ambiente.

(a) CO-AF (b) SprintIR (c) LuminOX

Figura 3.3: Sensores de concentración de gases

Los sensores utilizados fueron montados dentro de una atmósfera parcialmente aislada para poder realizar mediciones de prueba de las concentraciones de gases y las condiciones ambientales que se encuentran dentro de dicha atmósfera.

3.2.1 Circuitos controladores

Los sensores utilizados en este sistema cuentan con un circuito de control integrado o en placa, a excepción del sensor de monóxido de carbono CO-AF, el cuál se utiliza con el controlador EC200, mostrado en la Figura 3.4.

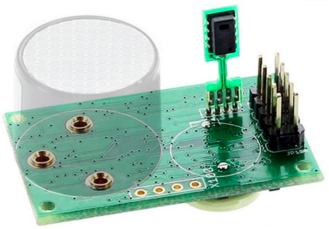


Figura 3.4: Controlador de sensor electroquímico EC200 [26].

Los circuitos controladores, además se ser la interfaz de comunicación de los sensores, sirven para acondicionar la señal análoga producida por la exposición de los sensores al gas seleccionado, convirtiendola en una señal digital, compensandola con los sensores de temperatura y presión y filtrandole el ruido existente. De este modo, los controladores ayudan a reducir el error y mejorar la precisión de las mediciones, además de que al transmitirse de forma digital, las señales transmitidas presentan una mayor inmunidad al ruido, lo que mejora la fiabilidad de la transmisión.

3.2.2 Configuración de los sensores

Para utilizar los sensores en la adquisición de datos es necesario realizar ciertas configuraciones. Éstas se realizan en el programa de adquisición, antes de capturar las mediciones, enviando comandos correspondientes para cada opción configurada en los sensores.

Para evitar la transmisión innecesaria de mediciones por los sensores y tener un mejor control del tiempo en el que se realiza el muestreo de las variables medidas, los sensores se configuran de forma que sólo transmitan las mediciones al ser solicitadas, de esta forma, la *Beaglebone* puede enviar un comando para solicitar las mediciones a los sensores en el momento en el que estas sean requeridas. Esta configuración se hace enviando los comandos **"`K 2`"** a los sensores ***SprintIR*** y ***CO-AF***, y **"`M 1`"** al sensor ***LuminOX***, con los cuales, los sensores entran en un modo llamado ***Polling Mode***.

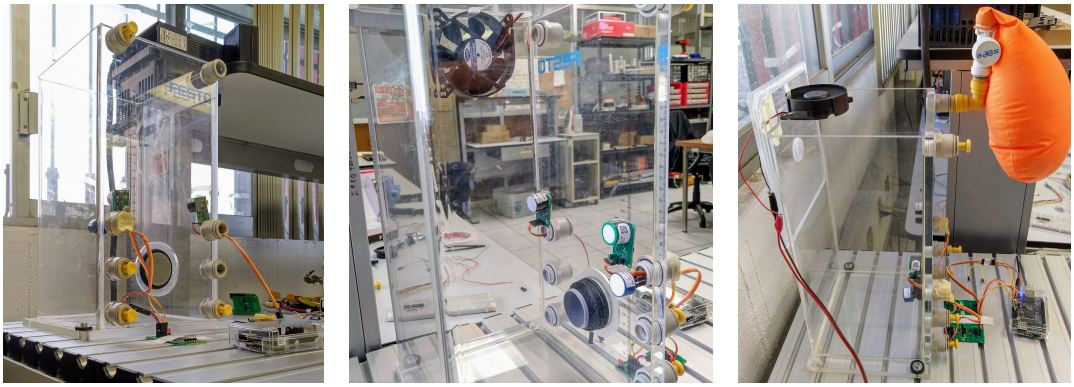
Con la finalidad de reducir la cantidad solicitudes de mediciones a los sensores y disminuir el tiempo de comunicación durante el muestreo de las variables, los sensores también se configuran de forma que transmitan todas las mediciones que serán requeridas para la adquisición a la vez, y así no tener que enviar un comando para obtener cada medición. Para realizar esta configuración se envía un comando específico a los sensores ***CO-AF*** y ***SprintIR*** dependiendo de las mediciones necesarias. El sensor ***CO-AF*** se configura con el comando **"`M 12358`"** para transmitir mediciones de concentración de monóxido de carbono, con y sin filtrado, temperatura, presión y humedad relativa, mientras que el sensor ***SprintIR*** se configura con el comando **"`M 6`"** para transmitir mediciones de concentración de bióxido de carbono con y sin filtrado. Por otra parte, el sensor ***LuminOX***, al contar ya con un comando para transmitir todas las mediciones necesarias en la adquisición (porcentaje y concentración de oxígeno, temperatura y presión), no requiere de alguna configuración adicional.

Para solicitar las mediciones que son configuradas en los sensores ***CO-AF*** y ***SprintIR*** se utiliza el comando **"`Q`"**, y para solicitar todas las mediciones del sensor *LuminOX*, el comando **"`A`"**. Los sensores responderán con una cadena donde se encuentran las mediciones solicitadas, identificando cada variable con un caracter clave.

3.3 Atmósfera parcialmente aislada

Para realizar las mediciones de prueba, y poder realizar algunas otras mediciones, se implementó una atmósfera con una caja de acrílico cerrada, como puede observarse en la Figura 3.5. En esta atmósfera pueden introducirse gases u objetos que produzcan alguna reacción química y así realizar mediciones de las concentraciones de monóxido y bióxido de carbono, oxígeno, temperatura, humedad y presión dentro de ella.

Esta atmósfera aislada cuenta con una válvula de control, con la cuál puede controlar el flujo de los gases que se van a medir. Cuenta con una tapa en la parte superior que puede ser retirada con facilidad para introducir o retirar objetos, o ventilar el interior de la atmósfera. También utiliza un ventilador como mezclador de gases, el cuál ayuda a mover los gases en el interior haciendo una mezcla más homogénea. Esta caja de acrílico también cuenta con varios orificios, por lo que es posible agregar más válvulas u otras conexiones para realizar pruebas o experimentos diferentes.



(a) Caja de acrílico (b) Montaje de los sensores (c) Atmósfera abierta

Figura 3.5: Atmósfera aislada utilizada para las mediciones de prueba.

3.4 Circuito de aislamiento

Ya que los sensores utilizados en este sistema pueden estar expuestos a distintas condiciones poco favorables dentro de la atmósfera, como humedad condensada, éstos son propensos a algún fallo que podría provocar un corto circuito o un sobrevoltaje en los sensores. Por esa razón se considera necesario proteger a la tarjeta la tarjeta *Beaglebone* y así evitar que en alguna situación extrema ésta se dañe y así correr también el riesgo de perder la información que se encuentra en la memoria de ésta.

Para hacer la conexión entre los puertos UART de la tarjeta *Beaglebone* y los sensores de forma segura se hace uso de aisladores digitales, los cuales son capaces de transmitir las señales de comunicación utilizando capacitores de aislamiento. Los aisladores digitales utilizados en este sistema son los circuitos integrados ISOW7842 del fabricante *Texas Instruments*, que cuenta con cuatro canales, un par para enviar señales y otro para recibir, como se muestra en Figura 3.6; con los cuales se pueden aislar hasta dos puertos UART. Ya que este sistema utiliza un puerto UART para cada sensor, es necesario de dos circuitos integrados como este para aislar tres puertos UART.

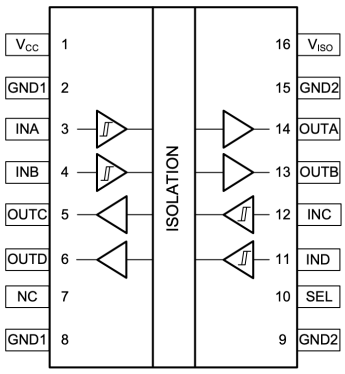
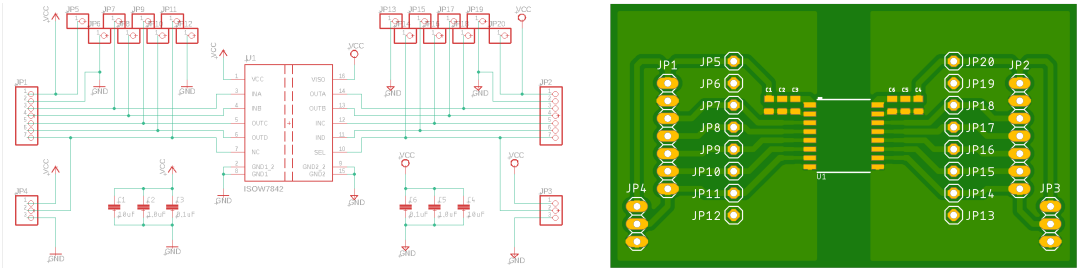


Figura 3.6: Diagrama de configuración de canales del ISOW7842[29].

El ISOW7842, cuenta con alimentación aislada que permite suministrar hasta 650 mW de potencia, por lo que no es necesario aumentar la cantidad de fuentes de CD para proveer de corriente a los sensores, o algún otro circuito en caso de ser necesario. A su vez, este circuito se alimenta con la misma

fuente de DC que provee energía a la *Beaglebone*.

Para utilizar el ISOW7842 se elaboró una tarjeta basada en el diseño de referencia de *Texas Instruments*[30], colocando el aislador digital ISOW7842, capacitores de desacople, pines para conexiones y puntos de prueba, como se muestra en la Figura 3.7(b). El esquemático del circuito implementado también se puede observar en la Figura 3.7(a), pudiéndose visualizar las conexiones de este circuito.



(a) Esquemático (b) Circuito impreso

Figura 3.7: Tarjeta de aislamiento de señales digitales.

El proyecto del diseño del PCB para el aislador digital se realizó en el *software Eagle* de *Autodesk*. Este proyecto está alojado en un repositorio público, el cuál puede ser accedido a través del enlace `https://gitlab.com/sayeth.rd/isow7842_testing_board`. El repositorio también puede ser clonado (descargado con todo su historial) libremente en un ordenador propio utilizando el comando `git clone` y la dirección web del repositorio desde la terminal.

En la Tabla 3.1 se muestran los pines de conexión de la tarjeta de aislamiento.

Tabla 3.1: Descripción de los pines de los conectores

Pin	Nombre	Descripción
JP1-1, JP4-1	V_{CC}	Voltaje de alimentación (entrada)
JP2-2, JP4-3	GND1	Tierra del voltaje de alimentación
JP1-3	INA	Entrada del canal A
JP1-4	OUTC	Salida del canal C
JP1-5	INB	Entrada del canal B
JP1-6	OUTD	Salida del canal D
JP1-7, JP4-2	NC	Sin conectar
JP2-1, JP3-1	V_{ISO}	Voltaje de alimentación aislado (salida) determinado por SEL
JP2-2, JP3-3	GND2	Tierra del voltaje de alimentación aislado
JP2-3	OUTA	Salida del canal A
JP2-4	INC	Entrada del canal C
JP2-5	OUTB	Salida del canal B
JP2-6	IND	Entrada del canal D
JP2-7, JP3-2	SEL	Selección de V_{ISO}. Si se conecta a GND2, entonces $V_{ISO} = 3.3\,V$. Si se conecta a V_{ISO}, entonces $V_{ISO} = 5\,V$.

3.4.1 Conexiones

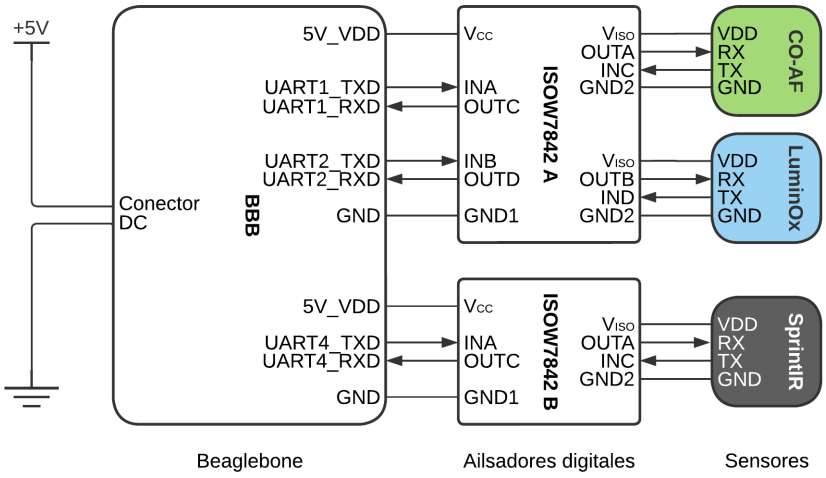


Figura 3.8: Diagrama de conexiones del sistema desarrollado.

La Figura 3.8 muestra la conexión de los sensores de concentración de gases a través de las tarjetas de aislamiento. El sensor *CO-AF* se conecta al *UART1*, el *LuminOx* al *UART2* y el *SprintIR* al *UART4*. Respetar este orden es crucial para el correcto funcionamiento del sistema debido a la diferencia en comandos y respuestas de cada sensor. Dado que cada tarjeta de aislamiento puede aislar sólo dos sensores, se requieren dos tarjetas para los tres.

La Figura 3.8 también ilustra la forma de alimentar cada etapa del sistema. La tarjeta *BeagleBone* se alimenta de la fuente de *5V*, las tarjetas de aislamiento se alimentan de la salida *5V_VDD* de la *BeagleBone* y los sensores se alimentan de la salida V_{ISO} de los circuitos de aislamiento; de esta forma, tanto las señales como la alimentación entre la *Beaglebone* y los sensores logra aislarse.

En la Tabla 3.2 se muestran los pines de la *Beaglebone* utilizados para conectar los puertos UART y la alimentación.

Tabla 3.2: Conexiones de la BeagleBone Black

Nombre	Pin	Descripción
5V_VDD	P9-5 ó P9-6	Alimentación (5V)
GND	P9-1 ó P9-2	Tierra (GND)
UART1_TXD	P9-24	Transmisión UART1 (TXD)
UART1_RXD	P9-26	Recepción UART1 (RXD)
UART2_TXD	P9-21	Transmisión UART2 (TXD)
UART2_RXD	P9-22	Recepción UART2 (RXD)
UART4_TXD	P9-13	Transmisión UART4 (TXD)
UART4_RXD	P9-11	Recepción UART4 (RXD)

3.5 Programa de adquisición de datos

Para la realizar la adquisición de datos con la *Beaglebone*, se desarrolló un programa escrito en lenguaje **C**. En este programa se realizan las tareas necesarias para la adquisición, tales como el manejo de los sensores a través de los puertos UART, y el registro de las mediciones.

Este programa debe ejecutarse desde una terminal de *Linux* en la *Beaglebone*, pudiendo acceder a ella a través del protocolo SSH. Para hacerlo es necesario ubicarse en el directorio donde está alojado el programa y ejecutar el comando `./daq <duración de la adquisición> <periodo de muestreo>`, sustituyendo los valores `<duración de la adquisición>` por el tiempo (en horas) durante el cuál se quiere realizar la adquisición, y `<periodo de muestreo>` por el tiempo deseado entre cada medición (en segundos). Si es necesario detener la adquisición antes de la finalización de su período establecido, se puede utilizar la combinación de teclas `Ctrl + c` en la terminal.

El programa desarrollado y su depuración se realizó con la ayuda del IDE en linea *Cloud9*. Este es posible utilizar desde un navegador web, accediendo al servidor de la misma *Beaglebone*, para esto es necesario conocer la dirección IP de la *Beaglebone* (lo cuál se puede saber con el comando `ip addr`). En este entorno fue posible visualizar los archivos del proyecto, depurar el código linea por linea, probar distintas funciones y observar las variables del código y de la estructura `termios`, entre otras cosas.

El programa realiza varias tareas para la adquisición de datos, específicamente estas tareas son: configurar los sensores, y a su vez los puertos UART para su utilización con éstos, crear el archivo de datos, registrando la fecha de inicio de la adquisición, solicitar mediciones a los sensores, recibir las mediciones e interpretarlas, para después registrarlas en el archivo de datos, y para finalizar, cerrar los archivos de los puertos UART, dejándolos así disponibles para cualquier otro uso. El código y sus funciones se explican a continuación.

3.5.1 Función principal

En la función `main()` del programa se realizan las tareas principales para la adquisición de datos de forma secuencial, como se muestra en el diagrama de la Figura 3.9. Esta función se encuentra en el archivo *main.c* de este proyecto, del cuál se muestra su código completo en Código A.1.

En esta secuencia de tareas, primero se configuran los sensores y los puertos UART haciendo uso de la función `sensConf()`, la cuál se describe más adelante en la Subsección 3.5.2. Con la ayuda de esta función se establecen configuraciones como: el número del puerto que se va a utilizar, los baudios de la comunicación, su modo de operación y las mediciones que tomarán los sensores.

Siguiendo el diagrama de flujo de la Figura 3.9, después de configurar los sensores se inicia la adquisición de datos, solicitando las mediciones a los sensores y registrandolas en un archivo de datos creado por el programa. Esta acción es representada por el bloque amarillo nombrado .ªdquisición de datosz se realiza con la función `DAQ()`, la cuál se describe a detalle en la Subsección 3.5.3. El muestreo de mediciones se realiza dentro de un bucle durante el tiempo establecido como "Duración de la adquisiciónz se realiza cada cierto tiempo, nombrado como "Periodo del muestreo".

Una vez que adquisición de datos finaliza, los archivos utilizados para acceder a los puertos UART se cierran para dejar éstos disponibles para otros programas. Para realizar esta acción se utiliza la función `uartClose()` explicada más adelante en la Subsección 3.6.2. Finalmente, el programa termina al retornar un valor de cero. En esta función también se va mostrando el progreso del programa, notificando en la terminal la ejecución de cada tarea, así como también las últimas mediciones tomadas.

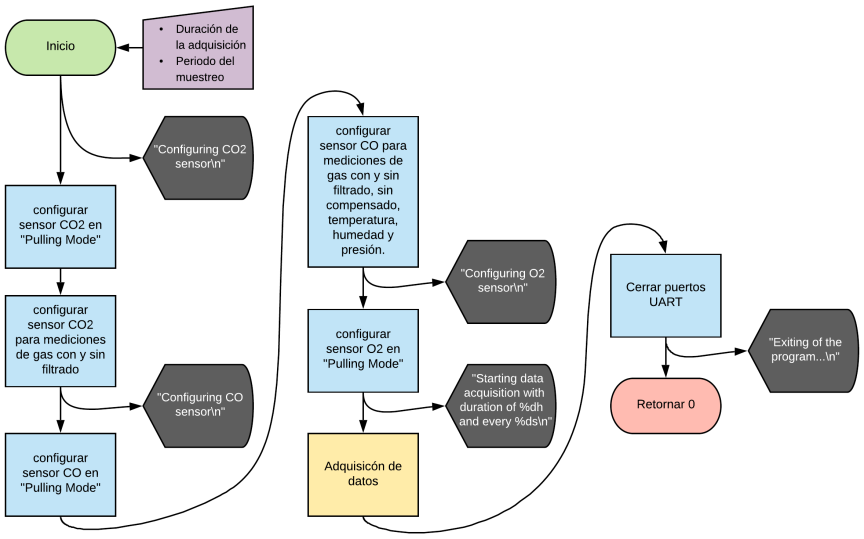


Figura 3.9: Diagrama de flujo general de la función `main` del programa de adquisición de datos.

3.5.2 Función `sensConf()`

Los procesos donde se realizan configuraciones en los sensores, mostrados en la Figura 3.9, para cambiar su modo de operación, y las mediciones que van a transmitir, son realizados con la función `sensConf`. Con esta función se pueden configurar algunas opciones permitidas por los sensores a través de comandos que se envían por un puerto UART. Su diagrama de flujo se muestra en la Figura 3.10.

Esta función tiene varios argumentos de entrada, los cuales pueden modificarse para ajustarlos a la configuración de cada sensor. Estos se explican a continuación:

- `uint8_t uartNumber`: Número de puerto UART que se utilizará para el sensor.
- `int baudRate`: Velocidad que se va a utilizar en la transmisión.
- `char mode[]`: Comando de configuración enviado al sensor.
- `char response[]`: Respuesta de confirmación esperada del sensor.
- `int tries`: Número de intentos para configurar el sensor en caso de alguna falla.

En esta función (véase Figura 3.10, para configurar el sensor, primero se configura el puerto UART a utilizar. Después, verifica si aún hay intentos restantes para intentar configurar el sensor. En caso de haberlos, se envía el comando de configuración y se espera la respuesta del sensor. Si se recibe la respuesta esperada, la función termina retornando el valor de `0`. De lo contrario se reducirá la cantidad de intentos y se volverá a intentar configurar el sensor. En caso de no haber intentos restantes, la función terminará enviando el valor de (`-1`), indicando que el sensor no pudo configurarse.

En esta función para configurar el puerto UART, transmitir y recibir datos, se hace uso de las funciones `uartConf`, `uartTransmit` y `uartReceive` de la librería `uart.h`. Para esperar a que los datos sean transmitidos, se hace uso de la función `tcdrain`, de la librería `termios.h`, y para mostrar los mensajes en la terminal se utiliza `printf`. En Código 3.5 se muestra la función `sensConf` completa.

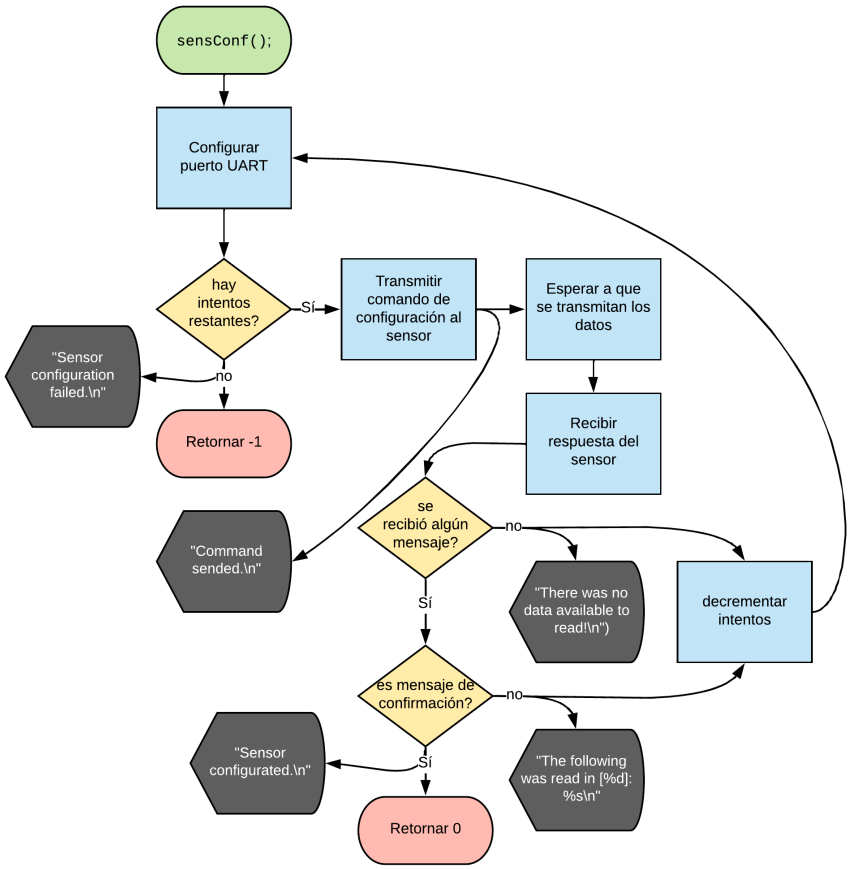


Figura 3.10: Diagrama de flujo la función `sensConf` del programa de adquisición de datos.

Código 3.5: Función `int sensConf(uint8_t uartNumber, int baudRate, char mode[], char response[], int tries)`.

```
1  int sensConf(uint8_t uartNumber, int baudRate, char mode[], char response[], int tries)
2  {
3          int count;
4
5          uartConf(uartNumber, baudRate);
6          while(tries){
7                  uartTransmit(uartNumber, mode);
8                  tcdrain(uartFile[uartNumber]); //wait all data has been sent
9                  printf("Command sended.\n");
10                 count = uartReceive(uartNumber);
11                 if (count == 0) printf("There was no data available to read!\n");
12                 else if (strcmp(receive[uartNumber], response) == 0) {
13                         printf("Sensor configurated.\n");
14                         return 0;
15                 } else {
16                         printf("The following was read in [%d]: %s\n",count,receive[uartNumber]);
17                 }
18                 tries --;
19         }
20         printf("Sensor configuration failed.\n");
21         return -1;
22 }
```

3.5.3 Función de adquisición de datos `DAQ()`

En la función `DAQ`, mostrada en la Figura 3.11, se realizan las tareas para tomar las muestras de las mediciones. En esta función se crea el archivo de datos, se solicitan las mediciones a los sensores, se reciben y se almacenar en el archivo de datos. La función utiliza los siguientes parámetros de entrada:

- `int t_hrs`: Establece la cantidad de horas durará la adquisición de datos.
- `int sp_s`: Establece la cantidad de segundos entre cada muestra.

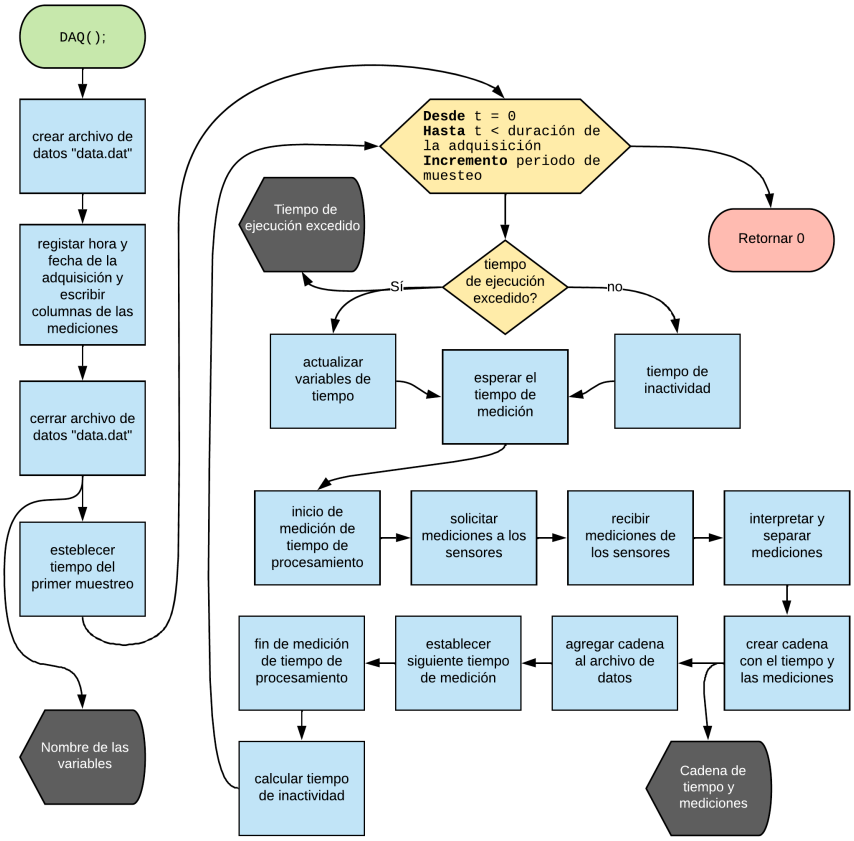


Figura 3.11: Diagrama de flujo la función `DAQ` del programa de adquisición de datos.

Esta función comienza creando el archivo "`data.dat`", donde se guardaran los datos, en el mismo directorio donde está alojado el proyecto. Posteriormente, registra la hora y fecha en la que se comenzó la adquisición en la primera linea del archivo, y debajo escribe el encabezado de las variables medidas en la adquisición con los nombres de cada una (tiempo, concentraciones de gases, temperatura, etc.) separados por una tabulación horizontal (TVS) para crear las columnas. Este encabezado también se muestran en la terminal para tener más claridad al ser monitoreadas. Estas acciones mencionadas se realiza con las lineas mostradas en la Sección de Código 3.6.

Código 3.6: Comienzo de la funcinón `int DAQ(int t_hrs, int sp_s)`.

```
int DAQ(int t_hrs, int sp_s)
{
        FILE* dfp; // create a file pointer
        const char data_header[200] =
                "t\tCO2\tCO2_f\tCO\tCO_f\tO2\tO2\tT_CO\tT_O2\tP_CO\tP_O2\tRH\n"\
                "s\tppm\tppm\tppm\tppO2\t%\tC*10\tC\tmBar\tmBar\t%";
        //measurements varirables
        char co2_uf[10]="",     co2_f[10]="",
                co_uf[10]="",   co_f[10]="",
                o2_ppm[10]="",  o2_xcent[10]="",
                co_temp[10]="", o2_temp[10]="",
                co_press[10]="",o2_press[10]="",
                co_relH[10]="", DATA[200]="";
        time_t curtime; //current time (date)
        clock_t start_t, end_t; //processing time measurements variables
        time_t next_samp_time, t0; //time control variables
        //maximum cycle iteration time
        double iteration_time_ms = sp_s*1e6 - 0.2e6;
        double inactivity_time = 1; //time to sleep

        time(&curtime); //saving date in curtime
        //write start data
        dfp = fopen(data_file_path, "w"); // open file for writing
        // send the value to the file
        fprintf(dfp, "Starting_DAQ_at_%s\n", ctime(&curtime));
        fclose(dfp); // close the file using the file pointer
        printf("%s", colums); //display variables colums
```

Después de haber creado el archivo, con la fecha y las columnas de cada medición, y habiendo establecido el tiempo del próximo muestreo, como se muestra en la Sección de Código 3.7, la función entra en un ciclo, que se repetirá para realizar cada muestreo hasta que termine el tiempo de adquisición. Este ciclo se controla con la variable "`t`", que comienza en cero y va aumentando el periodo de muestreo en cada iteración hasta que sobrepasa la duración de la adquisición, terminando con el ciclo.

Código 3.7: Parámetros del ciclo de la función DAQ y los tiempos de adquisición.

```
        next_samp_time = time(NULL)+1;//setting next sampling time
        t0 = next_samp_time;//saving initial time
        //cycle from 0 to adquisition time (seconds), incrementing sampling period
        for(time_t t = 0; t < (t_hrs*3600); t+=sp_s){
```

Dentro de este ciclo se envían los comandos de solicitud de mediciones a los sensores y se reciben las respuestas de estos mismos, las cuales se interpretan y, posteriormente, se extraen de estas las mediciones de cada variable y se almacenan en cadenas (de caractéres) individuales. Con estas cadenas se construye una cadena, con el tiempo y todas las mediciones realizadas, separados por una tabulación horizontal, que se muestra en la terminal, para que las mediciones puedan ser monitoreadas, y se agrega al final del archivo de datos "`data.dat`". Finalmente, se establece el tiempo del próximo muestreo para la siguiente iteración del ciclo. Todas estas tareas son realizadas con la sección de código mostrada en Código 3.8.

Código 3.8: Tareas realizadas en el ciclo de la función DAQ para solicitar, obtener, mostrar y almacenar las mediciones.

```
        start_t = clock(); //saving start time
          //transmiting commands to sensors
          uartTransmit(COAF, get_readigns);
          uartTransmit(LOX_O2, Readings_OX);
          uartTransmit(SprintIR, get_readigns);
          uartReceive(COAF); //receiving replys from
          uartReceive(LOX_O2);
          uartReceive(SprintIR);
          //interpreting and spliting measurements in variables
          memcpy(co2_uf,  getMeasures(receive[SprintIR],'z', 5), 5);
          memcpy(co2_f,   getMeasures(receive[SprintIR],'Z', 5), 5);
          memcpy(co_uf,   getMeasures(receive[COAF],    'z', 5), 5);
          memcpy(co_f,    getMeasures(receive[COAF],    'Z', 5), 5);
          memcpy(o2_ppm,  getMeasures(receive[LOX_O2],  'O', 6), 6);
          memcpy(o2_xcent,getMeasures(receive[LOX_O2],  '%', 6), 6);
```

```
16            memcpy(co_temp, getMeasures(receive[COAF],    'T', 5), 5);
17            memcpy(o2_temp, getMeasures(receive[LOX_O2],  'T', 5), 5);
18            memcpy(co_press,getMeasures(receive[COAF],    'B', 5), 5);
19            memcpy(o2_press,getMeasures(receive[LOX_O2],  'P', 4), 4);
20            memcpy(co_relH, getMeasures(receive[COAF],    'H', 5), 5);
21            //saving formated measurements in string DATA
22            sprintf(DATA,
23                    "%d\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s",
24                    (int)t, co2_uf, co2_f, co_uf, co_f, o2_ppm, o2_xcent,
25                    co_temp, o2_temp, co_press, o2_press, co_relH);
26            printf("\r%s", DATA); //showing measurements on display
27
28            dfp = fopen(data_file_path, "a"); // open file for writing
29            // saving measurements string to data file
30            fprintf(dfp, "%s\n", DATA);
31            fclose(dfp); // close the file using the file pointer
32            next_samp_time += sp_s; //set next sampling time
33          end_t = clock(); //saving end time
34          //calculate time to sleep
35          inactivity_time = iteration_time_ms - ((double)(end_t - start_t)*1e6 / CLOCKS_PER_SEC);
```

El tiempo de ejecución que le lleva al procesador hacer estas tareas, es medido en cada iteración, con el fin de saber el tiempo restante para siguiente muestreo y poner el programa inactivo durante este tiempo. La medición del tiempo de procesamiento se hace utilizando la función `clock()`, de la librería `time.h`, obteniendo la diferencia entre los ciclos de reloj antes y después de realizar las tareas, y convirtiéndola a microsegundos.

El tiempo de inactividad es calculado con el tiempo de procesamiento medido y el periodo de muestreo, considerando un margen a favor para que el administrador de tareas del sistema operativo regrese al programa y se realicen otras operaciones que no son consideradas en la medición del tiempo de procesamiento. Este tiempo de inactividad obtenido es verificado, por si la duración del muestreo anterior es excesiva y el tiempo de inactividad calculado resultó menor a 1 microsegundo (sin tiempo de inactividad). En este caso, se muestra un aviso en la terminal (indicando que el tiempo en la adquisición anterior ha sido excedido), se establece el siguiente muestreo en el próximo segundo y se actualiza el valor de tiempo de la adquisición, de lo contrario, si hay tiempo de inactividad, entonces se pone el programa en reposo. Después de manejar los dos casos, en los que hay o no tiempo de inactividad, se espera a que llegue el tiempo de muestreo como se muestra en la Sección de Código 3.9.

Código 3.9: Código para verificar el tiempo de inactividad y esperar el próximo muestreo.

```
1         if (inactivity_time <= 1){ //checking inactivity time
2                 usleep((int)inactivity_time); //inactivity
3         }
4         else{
5                 printf("Ejecution_time_exceded.\n%s", colums);
6                 next_samp_time = time(NULL)+1; //refreshing time values
7                 t = next_samp_time-t0;
8         }
9         //synchronizing/waiting to start measurements
10        while(next_samp_time != time(NULL));
```

El tiempo de cada muestreo se maneja con la función `time()`, de la librería `time.h`, con la cuál se obtiene la hora actual del calendario del sistema. De esta forma se puede establecer un tiempo de referencia (en el cuál inicia la adquisición) y saber cual es la hora actual para realizar los muestreos con una resolución de segundos.

En los muestreos, la solicitud y recepción de mediciones se hacen con las funciones `uartTransmit` y `uartReceive`, de la librería `uart.h`, enviando el comando de solicitud de mediciones correspondiente a cada sensor y recibiendo las mediciones en el *buffer* de recepción de cada puerto UART. Las mediciones se obtienen individualmente utilizando la función `getMeasures`, mostrada en la Sección de Código 3.10.

Código 3.10: Función char *getMeasures(char src[], char fval, int nchar).

```
1  char *getMeasures(char src[], char fval, int nchar)
2  {
3          char * ptr = &src[0];
4          static char s[10]="";
5
6          while(*ptr != fval) ptr++;
7          ptr += 2;
8          memcpy(s, ptr, nchar);
9          return s;
10 }
```

La función **getMeasures** recibe como argumentos los siguientes parámetros:

- `char scr[]`: Cadena de caractéres enviada por sensores, contiene las mediciones.
- `char fval`: Carácter que identifica cada tipo de medición.
- `int nchar`: Número de caractéres que tiene el valor de la medición.

Así mismo, la función regresa lo siguiente:

- `static char s`: Cadena de caractéres de la medición solicitada.

Esta función hace un escaneo en la cadena, desde el primer caracter, buscando el caracter de la medición que se quiere obtener (`fval`). Cuando lo encuentra, avanza hasta el valor de la medición (saltandose el espacio) y copia el valor de la medición en una cadena (`s`), la cuál retorna.

3.6 Libreria `uart.h v1.0`

Para realizar la comunicación, a través de los puertos UART de la *Beaglebone*, en el programa de adquisición de datos escrito en lenguaje **C**, es necesario crear una librería con las funciones para la comunicación a través de los puertos UART, la cuál es utilizada en el programa para tener interacción con los puertos y a su vez con los sensores. Esta librería se encuentra alojada en el repositorio del proyecto, el cuál puede ser accesado desde el enlace: https://gitlab.com/sayeth.rd/daq-serial-bbb.

La librería `uart.h` contiene las funciones:

- `uartConf`, configura el puerto UART para su utilización,
- `uartClose`, cierra el puerto una vez haya sido desocupado,
- `uartTransmit`, transmite datos a través de los puertos UART y
- `uartReceive`, recibe datos de los puertos UART.

Cada función es descrita a continuación.

3.6.1 Función `uartConf()`

La función `uartConf`, mostrada en la Sección de Código 3.11, es utilizada para configura los puertos UART, estableciendo los parámetros de entrada, control y salida de la comunicación. Esta función

permite elegir el puerto UART que se va a configurar y los baudios a los que operará, recibiendo estos parámetros como valores de entrada.

Para configurar el puerto, en la función, se abre el archivo del puerto, en el sistema de archivos del SO, y se asignan los atributos de comunicación del puerto a una estructura `termios` para poder manipularlos. Estos atributos de la comunicación de entrada, control y salida son limpiados para después asignar los deseados. De esta forma, se asignan los parámetros de control:

- la tasa de baudios recibida por la función
- mascara de los caractéres de ocho bits de tamaño
- receptor habilitado
- ignorar lineas de control del módem.

En los parámetros de entrada solamente se inhabilita la detección de errores de paridad. Posteriormente se limpia el *buffer* de transmisión, para descartar información no transmitida, y se aplican los cambios de las opciones del puerto UART.

Código 3.11: Función `int uartConf(uint8_t uartNumber, int baudRate)`.

```
1  int uartConf(uint8_t uartNumber, int baudRate)
2  {
3          char fullFileName[11];
4
5          //completing uart file path
6          if ((uartNumber==1)||(uartNumber==2)||(uartNumber==4)||(uartNumber==5))
7                  sprintf(fullFileName, incomplet_uart_path "%d", uartNumber);
8          else{
9                  perror("Wrong UART number. " \
10                 "UART numbers availables 1, 2, 4 or 5.\n");
11                 return UART_NUMBER_INCORRECT;
12         }
13
14         //openign uart file
15         printf("Configuring UART%d.\n", uartNumber);
16         if ((uartFile[uartNumber] = open(fullFileName, O_RDWR | O_NOCTTY | O_NDELAY | O_NONBLOCK))<0){
17                 perror("UART: Failed to open the file.\n");
18                 return -1;
19         }
20
21         //Sets the parameters associated with file
22         tcgetattr(uartFile[uartNumber], &options);
23         //cleaning flags
24         options.c_ispeed = 0;   options.c_lflag = 0;
25         options.c_line = 0;     options.c_oflag = 0;
26         options.c_ospeed = 0
27         // Set up the communications options:
28         options.c_cflag = baudRate | CS8 | CREAD | CLOCAL; // 8-bit, enable receiver, no modem control lines
29         options.c_iflag = IGNPAR; //ignore partity errors, CR -> newline
30         tcflush(uartFile[uartNumber], TCIOFLUSH); //discard file information not transmitted
31         tcsetattr(uartFile[uartNumber], TCSANOW, &options); //changes occur immmediately
32         printf("UART%d configurated.\n", uartNumber);
33
34         return UART_FUNCTION_SUCCESSFUL;
35 }
```

3.6.2 Función `uartClose()`

La función `uartClose`, mostrada en la Sección de Código 3.12, se utiliza para cerrar el puerto UART, una vez que haya sido desocupado por el programa, para que éste quede disponible para su

utilización por otro programa. Esta función recibe como parámetro de entrada el número del puerto UART que se desea cerrar.

Para cerrar el puerto UART, la función accede al archivo del puerto UART en el sistema de archivos del SO para cerrarlo con la función `close`, introduciendo la ruta del archivo UART que se quiere cerrar.

Código 3.12: Función `int uartClose(uint8_t uartNumber)`.

```
1  int uartClose(uint8_t uartNumber)
2  {
3      printf("Closing UART%d.\n", uartNumber);
4      close(uartFile[uartNumber]);
5      return UART_FUNCTION_SUCCESSFUL;
6  }
```

3.6.3 Función `uartTransmit()`

La función `uartTransmit`, mostrada en la Sección de Código 3.13, se utiliza para transmitir datos a través del puerto UART. Esta función recibe como parámetros de entrada: el número del puerto UART que se va a utilizar para transmitir los datos y el mensaje que va a ser enviado.

Para transmitir los datos, la función, una vez configurado el puerto, accede al archivo del puerto UART mediante el sistema de archivos del SO para escribir el mensaje en éste con la función `write`, introduciendo la dirección del archivo, el mensaje, y su longitud. Posteriormente, se descarta la información no transmitida para asegurar que el *buffer* de salida quede limpio.

Código 3.13: Función `int uartTransmit(uint8_t uartNumber, char message[])`.

```
1  int uartTransmit(uint8_t uartNumber, char message[])
2  {
3     int count;
4
5     //writing file
6     if ((count = write(uartFile[uartNumber], message, (strlen(message))))<0){       //send the string
7        perror("Failed to write to the output\n");
8        return -1;
9     }
10    tcflush(uartFile[uartNumber], TCOFLUSH);
11
12    return UART_FUNCTION_SUCCESSFUL;
13 }
```

3.6.4 Función `uartReceive()`

La función `uartReceive`, mostrada en la Sección de Código 3.14, se utiliza para obtener los datos recibidos a través del puerto UART. Esta función recibe como parámetros de entrada el número del puerto UART donde se reciben los datos.

Para obtener los datos recibidos, una vez ajustados los parámetros de comunicación, la función accede al archivo del puerto UART, mediante el sistema de archivos del SO, usando la función `read` para leerlo. Para usar esta función, se deben de introducir: la dirección del archivo, el puntero de la variable donde se guardarán los datos leídos y el número de bits que se van a leer. Los *buffers* de recepción de datos de los puertos UART son declarados de forma global en la librería `uart.h` de la siguiente forma: `char receive[6][100]`.

Código 3.14: Función `int uartReceive(uint8_t uartNumber)`.

```
1  int uartReceive(uint8_t uartNumber)
2  {
3      int count;
4
5      while((count = read(uartFile[uartNumber], (void*)receive[uartNumber], 100)) < 0);
6      tcflush(uartFile[uartNumber], TCIFLUSH);
7
8      return count;
9  }
```

3.7 Archivo de datos

El programa desarrollado para este sistema crea un archivo de datos, en el cuál se almacenan las mediciones obtenidas por los sensores. Este archivo de datos (`data.dat`) es creado en la carpeta "DATA", en el mismo directorio donde se encuentra el programa. En caso de existir un archivo con el mismo nombre en la carpeta, el programa lo sobre-escribirá con un archivo de datos nuevo. En este archivo se registra además la hora y fecha en la cuál se inicia la adquisición.

En el archivo se crean columnas para cada medición, separándolas con una tabulación horizontal, iniciando con la columna del tiempo de las mediciones y siguiendo con las mediciones de concentración de bióxido de carbono, monóxido de carbono, oxígeno, y en seguida las mediciones de las condiciones de la atmósfera temperatura, presión y humedad relativa;como se muestra en el Listado 3.15.

Código 3.15: Ejemplo del archivo `data.dat` con las mediciones obtenidas

```
1  Starting DAQ at Fri Feb 28 17:05:16 2020
2  t       CO2     CO2 f   CO      CO f    O2      O2      T CO    T O2    P CO    P O2    RH
3  s       ppm     ppm     ppm     ppm     ppO2    %       C*10    C       mBar    mBar    .%
4  0       00243   00236   00000   00000   0165.1  020.36  01254   +24.8   00811   0811    00382
5  1       00226   00238   00226   00000   0165.1  020.36  01254   +25.1   00811   0811    00382
6  2       00240   00238   00001   00000   0165.2  020.36  01254   +25.0   00813   0811    00382
7  3       00243   00236   00000   00000   0165.2  020.36  01254   +25.0   00814   0811    00382
8  4       00227   00240   00002   00000   0165.1  020.37  01254   +25.2   00814   0811    00382
9  5       00241   00242   00000   00000   0165.1  020.36  01254   +24.8   00814   0811    00382
10 6       00235   00243   00000   00000   0165.1  020.36  01254   +25.2   00814   0811    00382
```

En este archivo se almacenan las mediciones como son obtenidas de los sensores, por lo que conservan su formato original. Las mediciones de temperatura tomadas por la tarjeta controladora *EC200* (columna "T CO") tienen un desplazamiento de $+1,000$ y un factor de 10, por lo que para obtener las mediciones en grados Celsius hay que restar -1000 y después dividir el número entre 10 (e. g.$(01254 - 1000)/10 = 25.4$). Las mediciones de humedad relativa (columna "R H") también tienen un factor de 10, por lo tanto éstas se necesitan dividir entre 10 para poder ser interpretadas en porcentaje. Por otra parte, las mediciones de CO_2 (columnas "CO2" y "CO2 f") tienen un factor de 0.1, por lo que deben multiplicarse por 10 para obtener las mediciones en ppm.

El archivo de datos puede ser leído con el comando `cat` para visualizarlo completamente, o el comando `more` si se quiere leerlo progresivamente. También puede ser copiado a un dispositivo local con el comando `scp`.

3.8 Conclusiones

En este capítulo se detalló el desarrollo de un sistema embebido basado en BeagleBone Black para la adquisición de mediciones de gases y variables ambientales. El sistema, que incluye la BeagleBone,

sensores, un circuito de aislamiento y software de control en C, fue construido y probado preliminarmente. Se cumplió el objetivo de describir el desarrollo, incluyendo una librería UART reutilizable. Aunque funcional para pruebas de concepto, el sistema presenta limitaciones para entornos complejos. El siguiente capítulo evaluará el rendimiento a partir de los resultados de las pruebas.

WORK IN PROGRESS...

Capítulo 4

Pruebas y resultados

En este capítulo se presentan las pruebas realizadas con el sistema de adquisición de datos, con la finalidad de verificar el correcto funcionamiento del mismo, así como la validación de las mediciones obtenidas por éste. También se muestran los resultados de estas pruebas tanto de forma gráfica, como una tabla comparativa entre las diferentes repeticiones de cada tipo de prueba.

4.1 Pruebas

Para verificar el correcto funcionamiento del sistema de adquisición de datos desarrollado y la consistencia de las mediciones tomadas por éste, se realizan dos pruebas distintas, las cuales se repiten cinco veces para hacer una comparación entre los resultados obtenidos en cada prueba. En estas pruebas se miden las concentraciones de los gases oxígeno, monóxido y bióxido de carbono, así como la temperatura, humedad relativa y presión, dentro de la atmósfera aislada descrita en la Sección 3.3

Estas pruebas son: la medición de los gases de exhalación de una persona y la adquisición de datos del proceso de combustión de una vela, las cuales fueron propuestas por su repetibilidad, sencillez, y por ser fenómenos bien conocidos, respecto a posibles concentraciones esperadas. Éstas se detallarán a continuación.

4.1.1 Medición de los gases de exhalación de una persona

En esta prueba se realiza la medición de los gases exhalados por una persona al respirar profundamente, los cuales son almacenados primero en un globo, hasta reunir una cantidad considerable, para después introducirlos dentro de una atmósfera aislada, donde se realizan las mediciones por sensores instalados en ésta. En la Figura 4.1 se puede observar una ilustración de esta prueba.

El procedimiento de la realización de esta prueba es el siguiente. Primero se hace inhalar profun-

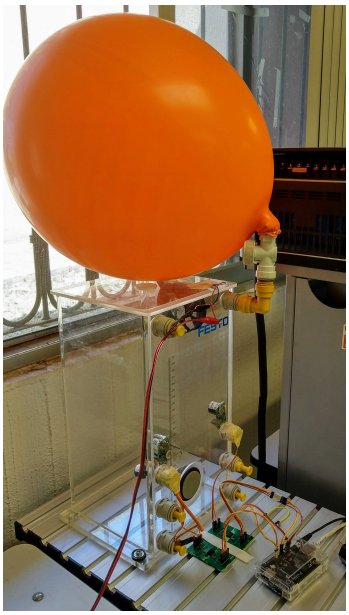


Figura 4.1: Globo conectado a la válvula de control de la atmósfera.

damente a una persona para después depositar sus gases de exhalación en un globo grande. Este paso se repite hasta que el globo consigue un volumen notablemente mayor al del recipiente de la atmósfera aislada, como se muestra en la Figura 4.2, para desplazar la mayoría del gas existente en el interior de ésta. Después, se prepara el inicio de las mediciones conectando el globo a la válvula de control de la atmósfera. Una vez conectado el globo, se inicia la adquisición de datos desde la terminal del sistema y se abre la válvula de control para introducir los gases a la atmósfera aislada. La válvula de control se cierra una vez que el globo quede sin presión. Luego de introducir los gases a la atmósfera, se espera hasta que las mediciones se estabilicen para después abrir y ventilar el recipiente retirando su tapa superior. La adquisición de datos es detenida una vez que los gases se estabilizan de nuevo.

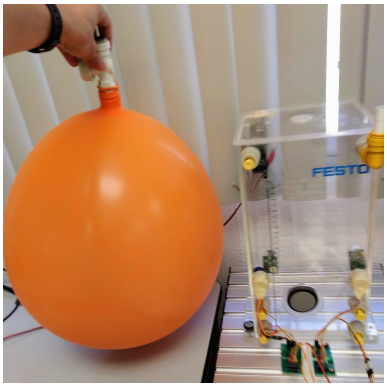


Figura 4.2: Globo inflado a lado de la atmósfera aislada

Los gases existentes en el interior de la atmósfera anteriores a la inserción de los gases de exhalación, son expulsados por la tapa superior del recipiente que, al sólo estar sobrepuesta, funciona como una válvula de escape.

4.1.2 Adquisición de datos del proceso de combustión

En esta prueba se adquieren las mediciones, ya mencionadas, durante la combustión de una vela dentro de la atmósfera asilada, como se muestra en la Figura 4.3, hasta que la falta de oxigeno extingue la llama de ésta.

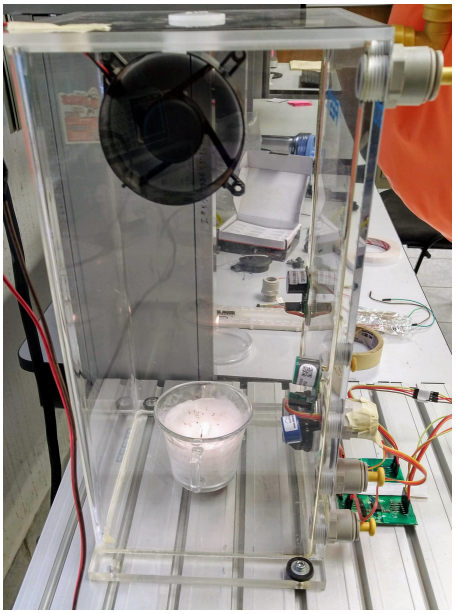


Figura 4.3: Vela haciendo combustión dentro de la atmósfera aislada.

Para realizar esta prueba se sigue el siguiente procedimiento. Primero, se coloca la vela apagada dentro del recipiente de la atmósfera aislada, dejando éste destapado. Una vez que la vela está colocada, se inicia la adquisición de datos y en seguida se enciende la vela. Después de ésto, se espera a que la llama se estabilice para posteriormente colocar la tapa superior del recipiente de la atmósfera aislada. En seguida, la llama de la vela se irá atenuando a medida que la concentración de oxígeno se reduce, hasta que ésta termina por extinguirse, lo que liberará una pequeña cantidad de humo en el interior de la atmósfera aislada. Posteriormente, se espera a las mediciones se estabilicen para después destapar la atmósfera y así mismo ventilarla. Finalmente, cuando las mediciones se estabilizan en los nuevos valores, la adquisición de datos termina.

4.2 Resultados de las mediciones de los gases de exhalación

En esta sección se presentan los resultados de las pruebas de medición de los gases de exhalación explicada anteriormente. Estos resultados se presentan tanto de forma gráfica, mostrando el comportamiento común de cada una de las variables en esta prueba, como con una tabla comparativa de los valores mínimos y máximos obtenidos de las variables medidas en cada repetición. También se presentan las mayores variaciones con respecto a la media que hubo de estos valores mínimos y máximos.

4.2.1 Comportamiento de las variables medidas

A continuación se presenta una serie de gráficas de las mediciones obtenidas durante esta prueba y una breve explicación del comportamiento de éstas, mostrando las mediciones de CO_2, en la Figura 4.4; O_2, en la Figura 4.5; humedad relativa, en la Figura 4.6; temperatura, en la Figura 4.7; presion, en la Figura 4.8; y finalmente CO, en la Figura 4.9. Estas gráficas corresponden a la primera prueba de medición de gases de exhalación realizada, nombrada como "prueba no. 1".

comentario para el revisor: a continuación se utilizarán saltos de página provisionales sólo para facilitar la lectura y redacción

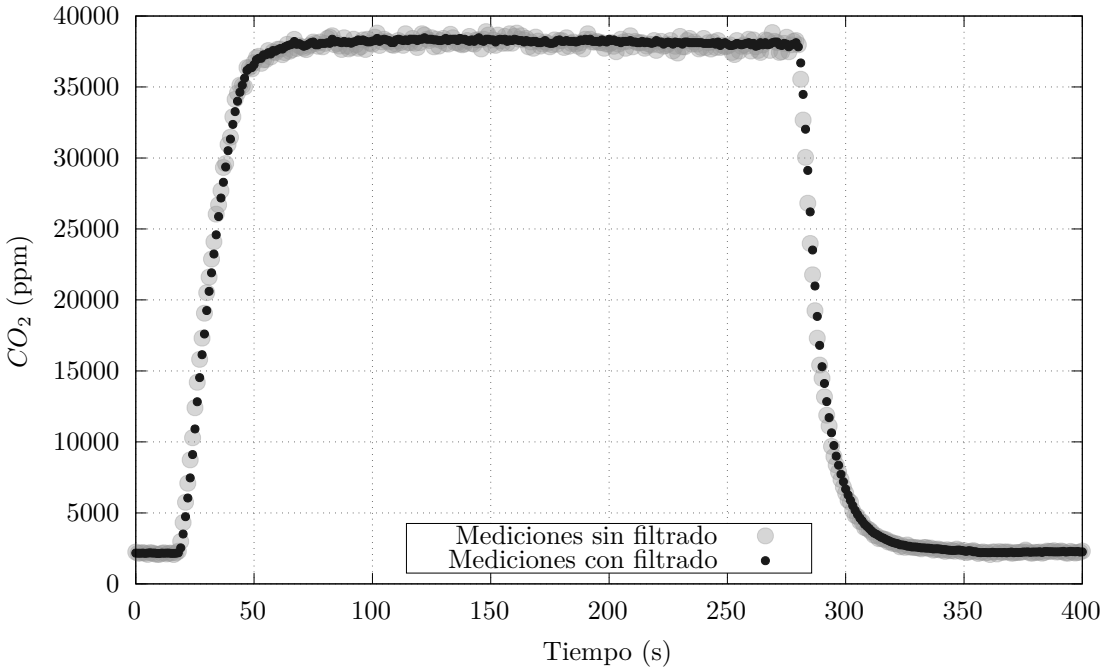


Figura 4.4: Gráfica de las mediciones de CO_2 dentro de la atmósfera durante la "prueba no. 1" de medición de gases de exhalación.

En la Figura 4.4 se muestra la gráfica de las mediciones de la concentración de CO_2 a través del tiempo, utilizando mediciones con y sin filtrado. En esta gráfica, se puede observar como las mediciones de CO_2 pasan de estar en un nivel bajo, casi estático, a un nivel alto en alrededor de 30 segundos debido a la entrada de los gases de exhalación a la atmósfera aislada. El nivel de concentración de CO_2 en las mediciones se mantienen en aproximadamente 40 mil ppm (4 % de concentración) durante más de 200 segundos, hasta que la tapa superior de la atmósfera es retirada. En ese momento las mediciones de concentración comienzan a caer de forma exponencial hasta alcanzar un nivel cercano al inicial en aproximadamente 70 segundos. Como puede notarse, las mediciones con filtrado son más uniformes y estáticas que las mediciones sin filtrado, pero también tienen una respuesta más lenta, la cuál se puede observar cuando la concentración de CO_2 incrementa, y cae.

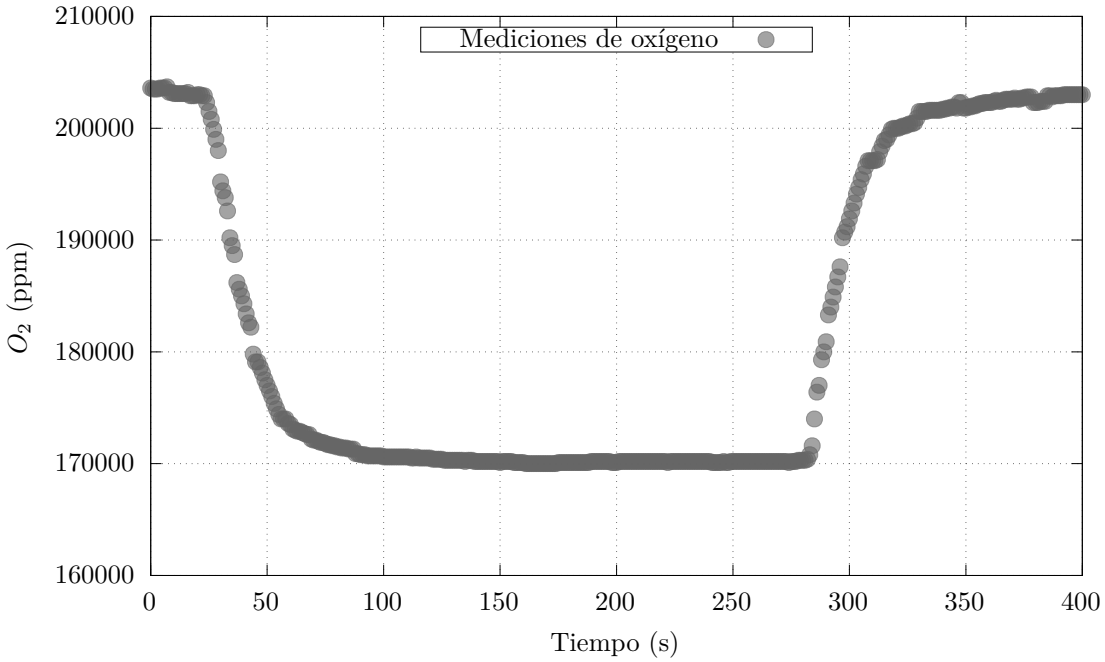


Figura 4.5: Gráfica de las mediciones de O_2 dentro de la atmósfera durante la "prueba no. 1" de medición de gases de exhalación.

En la Figura 4.5 se muestra la gráfica de la concentración de oxígeno (O_2) en partes por millón (ppm), durante la prueba de medición de gases de exhalación. En esta gráfica se observa como las mediciones de O_2 comienzan en un nivel superior a 200 mil ppm (20 %), y luego comienzan a decender abruptamente cuando se inicia a introducir los gases de exhalación, cerca del segundo 25, hasta que se estabilizan en al rededor de 170 mil ppm (17 %), aproximadamente en el segundo 100. A partir de ese momento, las mediciones se mantienen casi constantes, hasta que la atmósfera se comienza a ventilar cerca del segundo 280, es entonces cuando el nivel de concentración de oxígeno comienza a elevarse, recuperándose a un nivel cercano al inicial al final de la prueba (segundo 350).

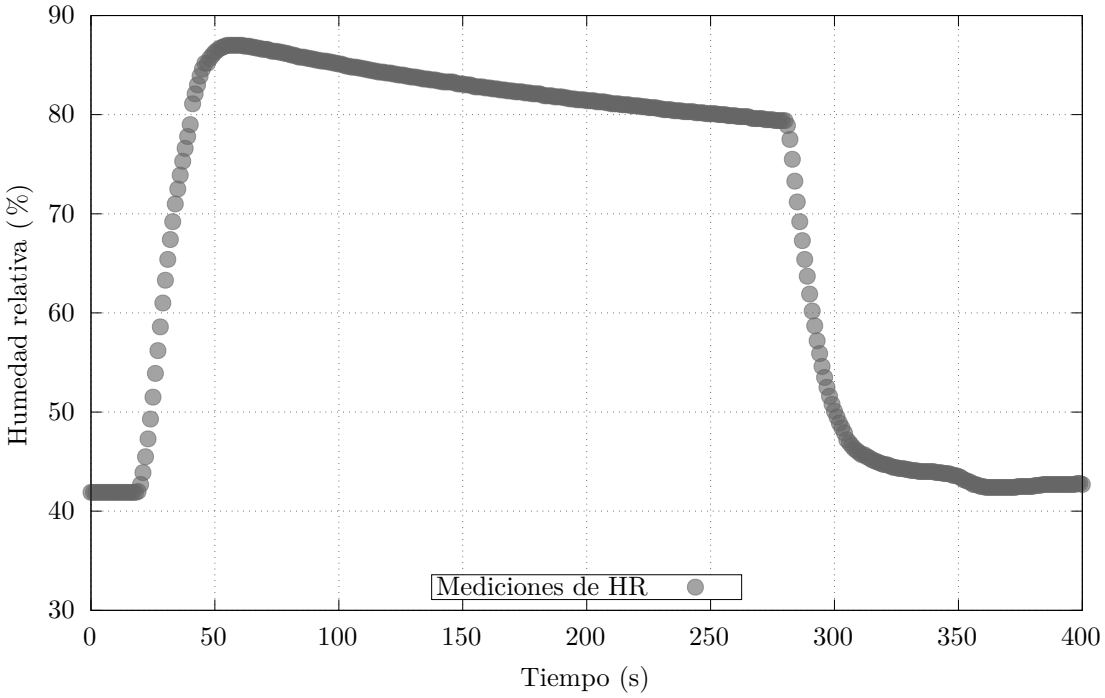


Figura 4.6: Gráfica de las mediciones de humedad relativa dentro de la atmósfera durante la "prueba no. 1" de medición de gases de exhalación.

En la Figura 4.6 se muestra la gráfica de las mediciones de la humedad relativa (HR) de esta prueba y su comportamiento. En la gráfica se observa como las mediciones de HR parten de un nivel de aproximadamente 42 % y comienzan a elevarse cuando se introducen los gases de exhalación, hasta llegar a su punto máximo en el segundo 50 aproximadamente. Después de esto, las mediciones comienzan a decender, cada vez más lentamente, hasta que la atmósfera es destapada y ventilada. En ese momento las mediciones caen abruptamente hasta un nivel cercano al inicial, quedando un remanente de humedad en la atmósfera.

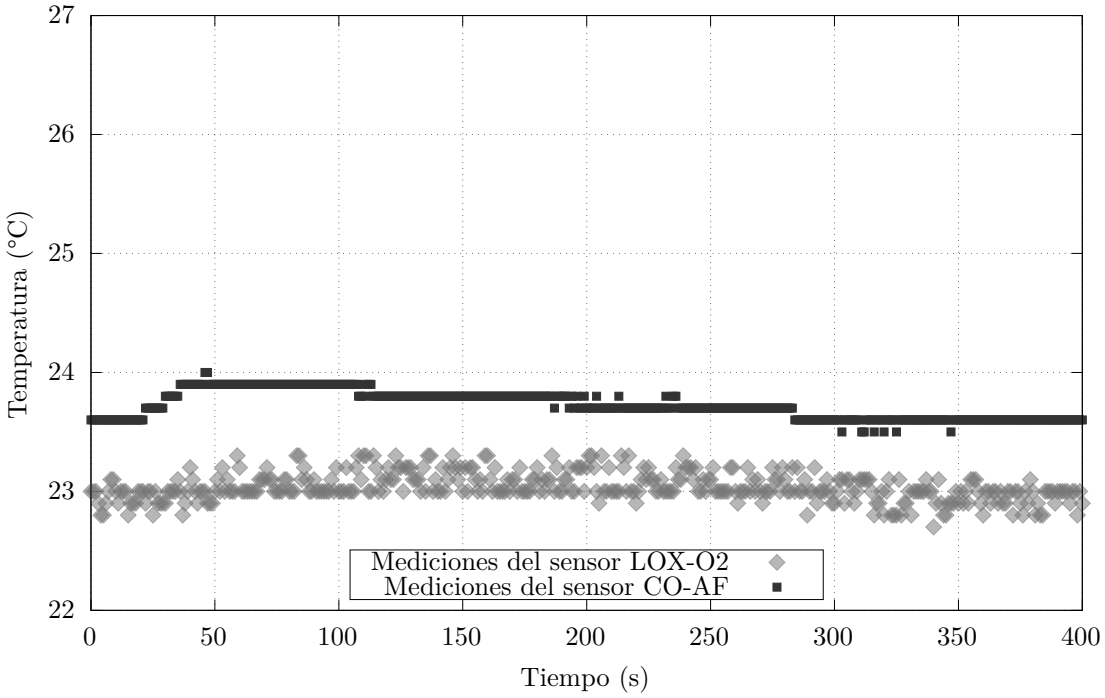


Figura 4.7: Gráfica de las mediciones de temperatura dentro de la atmósfera durante la "prueba no. 1" de medición de gases de exhalación.

En la Figura 4.7 se muestran las mediciones de temperatura realizadas en esta prueba por los sensores *LOX-02*, de oxígeno, y *CO-AF*, de monóxido de carbono. En la gráfica se puede observar que la temperatura tiene un valor inicial y esta aumenta levemente al introducir los gases de exhalación. Esta temperatura va disminuyendo lentamente con el paso del tiempo, habiendo un decremento levemente mayor al retirar la tapa superior de la atmósfera, hasta llegar a una temperatura cercana a la inicial al final de la prueba (segundo 350). En esta gráfica también puede notarse que el sensor *CO-AF* tiene una respuesta más rápida y una mejor estabilidad que el sensor *LOX-02*, además de que las mediciones obtenidas por el sensor *CO-AF* están aproximadamente 0.5 °C arriba de las del sensor *LOX-02*.

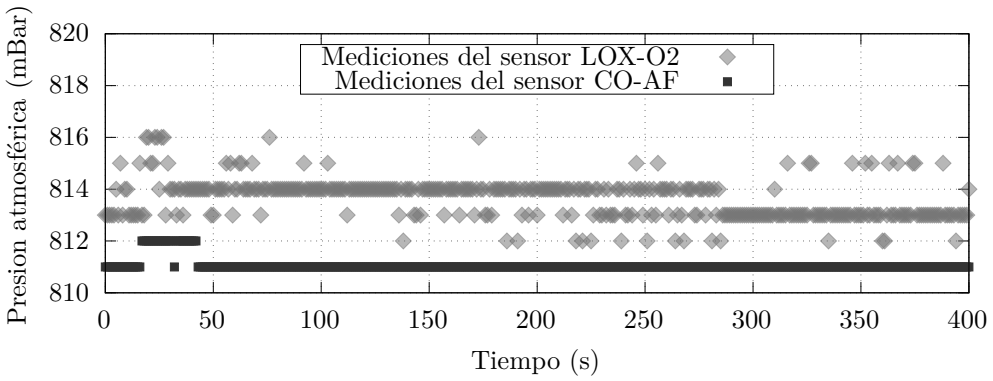


Figura 4.8: Gráfica de las mediciones de presión atmosférica dentro de la atmósfera durante la "prueba no. 1" de medición de gases de exhalación.

En la 4.8 se muestra la gráfica de las mediciones de presión en esta prueba y su comportamiento. En esta gráfica se puede observa que la presión tiene un leve incremento sólo en el momento de introducir los gases de exhalación, aproximadamente desde el segundo 20 hasta el 45. También puede notarse que la estabilidad de las mediciones presión del sensor *CO-AF* es mayor que la del sensor LOX-02.

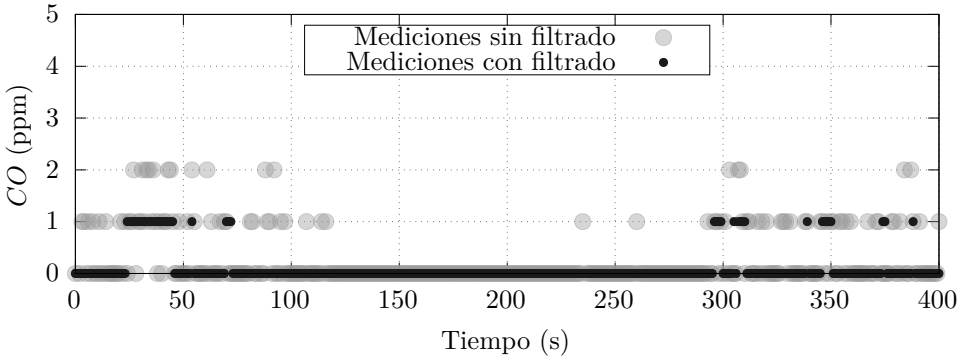


Figura 4.9: Gráfica de las mediciones de *CO* dentro de la atmósfera durante la "prueba no. 1" de medición de gases de exhalación.

El comportamiento del *CO* en la prueba de medición de gases de exhalación se muestra en la gráfica de la Figura 4.9. En esta gráfica se muestran las concentraciones tanto con filtrado como sin filtrado. Como puede observarse, las mediciones de *CO* se mantienen al mínimo, llegando hasta 1 ppm con filtrado y 2 ppm (0.0002 %) sin filtrado, ya que la exhalación de una persona no presenta concentraciones de *CO* considerables. También puede notarse que las mediciones sin filtrado llegan a tener una mayor variación comparado con las mediciones que sí lo tienen.

El comportamiento de las variables medidas en la prueba de mediciones de gases de exhalación se mantuvo similar en las repeticiones de esta prueba. Sus gráficas pueden ser observadas en la Sección A.3 de esta tesis.

4.2.2 Comparación de los resultados obtenidos de las pruebas de medición de gases de exhalación

En esta sección se presenta una comparación de los resultados obtenidos de las pruebas de medición de gases de exhalación, para analizar la consistencia del sistema este tipo de prueba. Por practicidad, está comparación está hecha con los valores mínimos y máximos de las variables medidas, obtenidos en todas las repeticiones de esta prueba. Las gráficas, donde se puede observar el comportamiento de las variables en las demás repeticiones de esta prueba, así como sus valores mínimos y máximos, pueden ser revisados en el Sección A.3 de esta tesis.

En la Tabla 4.1 se puede observar una comparación de los valores mínimos y máximos de las variables medidas en las cinco repeticiones de la prueba en las columnas centrales. En la primera columna se encuentran las variables medidas, indicando la fila para el valor mínimo y máximo de cada una de las pruebas. En la última columna se encuentra la media de los cinco valores mínimos y máximos obtenidos en cada prueba.

Los valores de las variables CO y CO_2 son tomados de las mediciones con filtrado, por su estabilidad ya vista en las gráficas del comportamiento de estos gases. De la misma forma, en las variables de presión y temperatura se utilizan las mediciones realizadas por el sensor de monóxido de carbono *CO-AF*.

Tabla 4.1: Tabla comparativa de valores mínimos y máximos de las variables medidas en varias pruebas de medición de los gases de exhalación.

Variables medidas		No. 1	No. 2	No. 3	No. 4	No. 5	Media
Concentración	Mín.	0	0	0	0	0	0.0
de ***CO*** (ppm)	Máx.	1	1	1	1	1	1.0
Concentración	Mín.	2,140	2,160	2,180	2,320	2,260	2,212
de $\boldsymbol{CO_2}$ (ppm)	Máx.	38,470	33,070	34,790	36,990	37,240	36,112
Concentración	Mín.	170,000	174,500	172,900	171,400	170,400	171,840
de $\boldsymbol{O_2}$ (ppm)	Máx.	203,800	203,600	203,300	203,900	203,200	203,560
Humedad	Mín.	41.9	40.9	41.9	37.8	39.2	40.34
relativa (%)	Máx.	87.0	87.0	86.3	82.5	83.0	85.16
Temperatura (°C)	Mín.	23.5	23.7	23.9	25.1	24.5	24.14
	Máx.	24.0	24.1	24.3	26.4	25.6	24.88
Presión (mBar)	Mín.	812	811	811	811	811	811.2
	Máx.	816	816	817	816	816	816.2

Como puede observarse en la tabla comparativa de las pruebas de medición de los gases de exhalación, los valores mínimos y máximos obtenidos de las mediciones tuvieron cierta consistencia. En los valores mínimos y máximos de la concentración de CO no hubo variación alguna.

Las mayores variaciones con respecto a la media de las demás variables se muestran en la Tabla 4.2, con la variación relativa a la media y el número de las pruebas en las cuales hubieron estas variaciones. En la primera columna de esta tabla se muestran las variables, las cuales tienen una fila para las variaciones de los valores mínimos y otra para las de los máximos. En seguida se muestran estas

variaciones, indicando la mayor variación inferior (desde), el número de prueba con esa variación, y la mayor variación superior (hasta), e igualmente el número de prueba.

Tabla 4.2: Tabla de variaciones con respecto a la media de los valores mínimos y máximos obtenidos de las pruebas de medición de los gases de exhalación.

		Variaciones			
Variables medidas		Desde	En la prueba	Hasta	En la prueba
Concentración	Mín.	-72 (-3.25 %)	No. 1	108 (4.88 %)	No. 4
de $\boldsymbol{CO_2}$ (ppm)	Máx.	-3,042 (-8.42 %)	No. 2	2,358 (6.53 %)	No. 1
Concentración	Mín.	-1,840 (-1.07 %)	No. 1	2,660 (1.55 %)	No. 2
de $\boldsymbol{O_2}$ (ppm)	Máx.	-360 (-0.18 %)	No. 5	340 (0.17 %)	No. 4
Humedad	Mín.	-2.5 (-6.30 %)	No. 4	1.6 (3.87 %)	No. 1 y 3
relativa (%)	Máx.	-2.7 (-3.12 %)	No. 4	1.8 (2.16 %)	No. 1 y 2
Temperatura (°C)	Mín.	-0.6 (-2.65 %)	No. 1	1.0 (3.98 %)	No. 4
	Máx.	-0.9 (-3.54 %)	No. 1	1.5 (6.11 %)	No. 4
Presión (mBar)	Mín.	-0.2 (-0.02 %)	No. 4	0.8 (0.10 %)	No. 1
	Máx.	-0.2 (-0.02 %)	No. 1, 2, 4 y 5	0.8 (0.10 %)	No. 3

Cómo puede observarse en la Tabla 4.2, las mayores variaciones se encuentran en los valores máximos de la concentración de CO_2, con diferencias desde -3,042 ppm hasta 2,358 ppm con respecto a la media de valores máximos obtenidos, representando un 8.42 % y 6.53 % de esta respectivamente. Las demás variaciones se encuentran por debajo del 6.5 % de la media de los valores obtenidos correspondientes, habiendo variaciones muy bajas como en los valores de la presión.

4.3 Resultados de las pruebas de combustión

En esta sección se presentarán los resultados de las pruebas de combustión, mostrando el comportamiento de las variables medidas durante la prueba con gráficas, y haciendo una comparativa de los valores mínimos y máximos de cada variable obtenidos en cada repetición de la prueba. También se presentan las mayores variaciones que hubo entre estos valores.

4.3.1 Comportamiento de las variables medidas en la prueba de combustión

A continuación se presenta una serie de gráficas de las mediciones obtenidas durante esta prueba y una breve explicación del comportamiento de éstas, mostrando las mediciones de CO_2, en la Figura 4.10; O_2, en la Figura 4.12; humedad relativa, en la Figura 4.13; temperatura, en la Figura 4.14; presion, en la Figura 4.15; y finalmente CO, en la Figura 4.11. Estas gráficas corresponden a la primera prueba de medición de gases de combustión, nombrada como "prueba no. 1".

El comportamiento de las variables medidas en la prueba de la adquisición de datos del proceso de combustión se observa en las gráficas presentadas a continuación. Estas gráficas corresponden a la primera repetición de esta prueba, llamada "prueba no. 1".

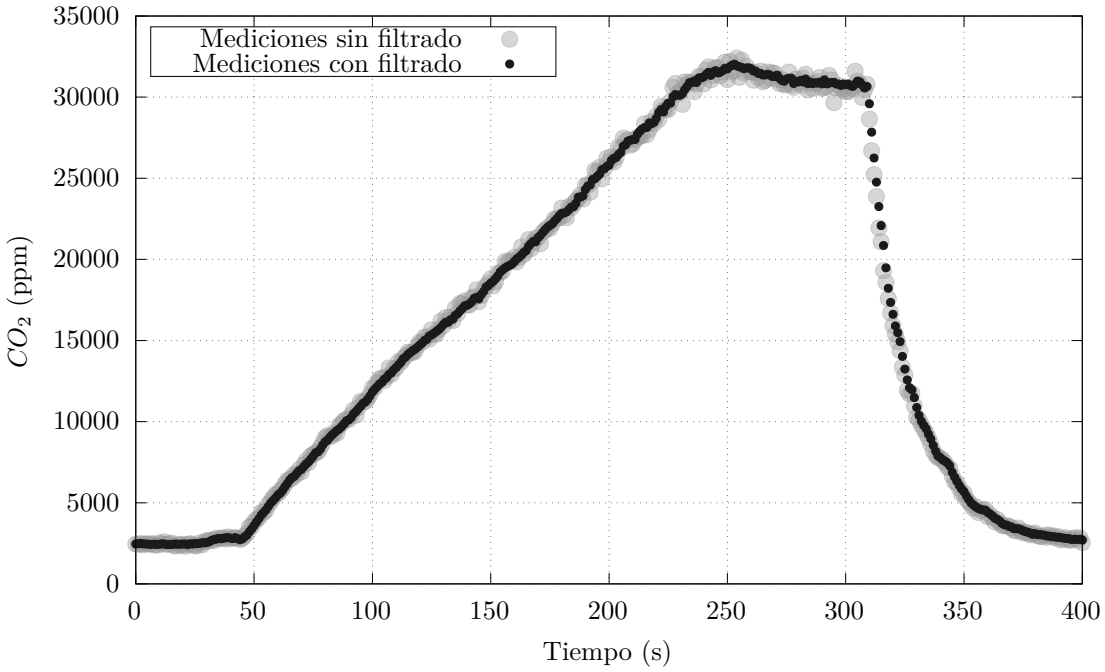


Figura 4.10: Gráfica de las mediciones de CO_2 dentro de la atmósfera durante la "prueba no. 1" de adquisición del proceso de combustión.

En la Figura 4.10 se muestra la gráfica de las mediciones con y sin filtrado de CO_2 obtenidas en esta prueba. En esta gráfica se observa como las mediciones comienzan en un nivel casi estático en un valor de 2,500 ppm (0.25 % de concentración), y después tienen una leve variación cuando se enciende la vela aproximadamente en el segundo 35. El valor de las mediciones comienza a incrementarse notablemente cuando la tapa superior de la atmósfera aislada es colocada, una vez que la flama de la vela se estabiliza antes de llegar el segundo 50. Este incremento se mantiene casi constante durante la combustión, hasta que la llama se extingue aproximadamente en el segundo 250, llegando a un valor de aproximadamente 32,500 ppm (3.25 % de concentración). Posteriormente, las mediciones de CO_2 tienen un leve decremento, y después del segundo 300 al destapar la atmósfera, hay un descenso abrupto, hasta que las mediciones se estabilizan en un valor cercano al inicial cuando la prueba finaliza (segundo 400). También puede notarse en está gráfica que las mediciones sin filtrado tienen una mayor variación y velocidad de respuesta que las mediciones con filtrado.

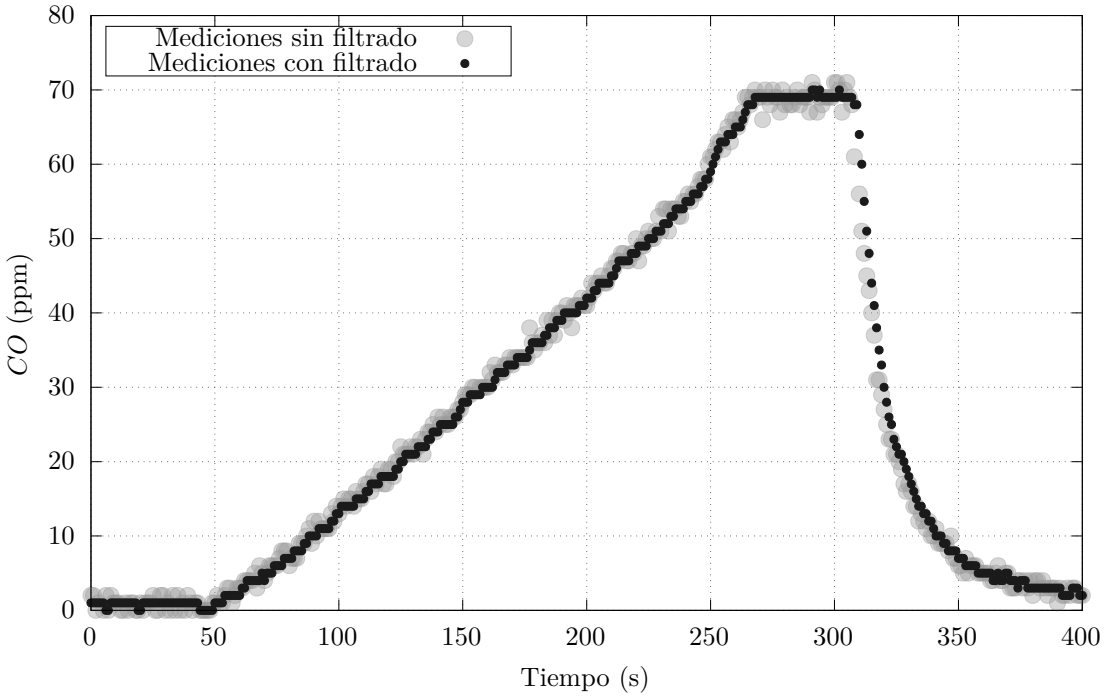


Figura 4.11: Gráfica de las mediciones de CO dentro de la atmósfera durante la "prueba no. 1" de adquisición del proceso de combustión.

En la Figura 4.11 se muestra la gráfica de las mediciones, con y sin filtrado, de CO obtenidas en esta prueba y su comportamiento. En esta gráfica se observa como el valor de las mediciones de la concentración de CO parten de casi cero, y comienzan a incrementar constantemente después de encenderse la vela y colocarse la tapa superior de la atmósfera aislada aproximadamente en el segundo 50. Después, las mediciones de CO aumentan aún más cuando la llama de la vela se extingue cerca del segundo 250, llegando casi 70 ppm (0.007 % de concentración). Las mediciones permanecen en este nivel, hasta que después del segundo 300 la atmósfera es destapada y ventilada. Entonces, el valor de las mediciones desciende drásticamente hasta llegar a cero ppm. Como se puede notarse en esta gráfica, las mediciones con filtrado son más estables, mientras que las mediciones sin filtrado tienen una respuesta más rápida.

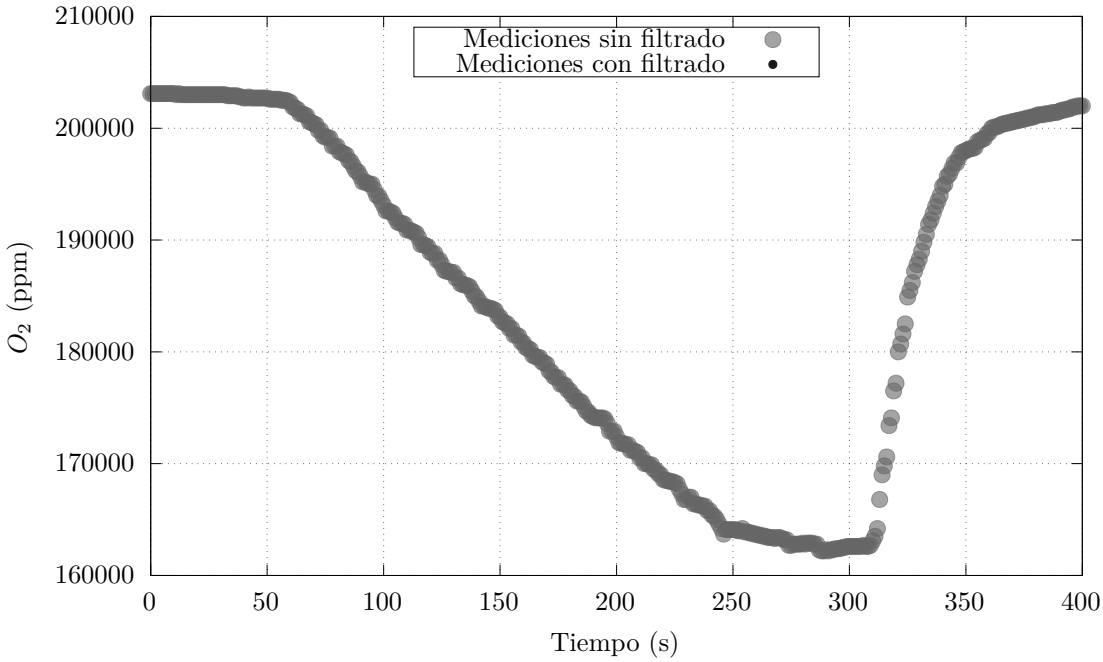


Figura 4.12: Gráfica de las mediciones de O_2 dentro de la atmósfera durante la "prueba no. 1" de adquisición del proceso de combustión.

En la Figura 4.12 se muestra la gráfica de las mediciones de la concentración de oxígeno en partes por millon (ppm) tomadas en esta prueba. En esta gráfica se observa como las mediciones de oxígeno tienen en un nivel superior a las 200 mil ppm (20 % de concentración) al iniciar la prueba. Estas mediciones comienzan a decender cuando la vela es encendida. Tiempo después de que la llama se estabiliza y la atmósfera es cerrada, al pasar el segundo 50, puede observarse que el decremento de oxígeno se acelera y sigue más o menos constante, hasta que desacelera cuando la llama de la vela se extingue pasando el segundo 250. La concentración de oxígeno disminuye a un mínimo de poco más de 160,000 ppm (16 % de concentración), y después, pasando el segundo 300, al destapar la atmósfera aislada, los valores de las mediciones de oxígeno se elevan abruptamente, hasta estabilizarse casi en el nivel inicial al final de la prueba (segundo 400).

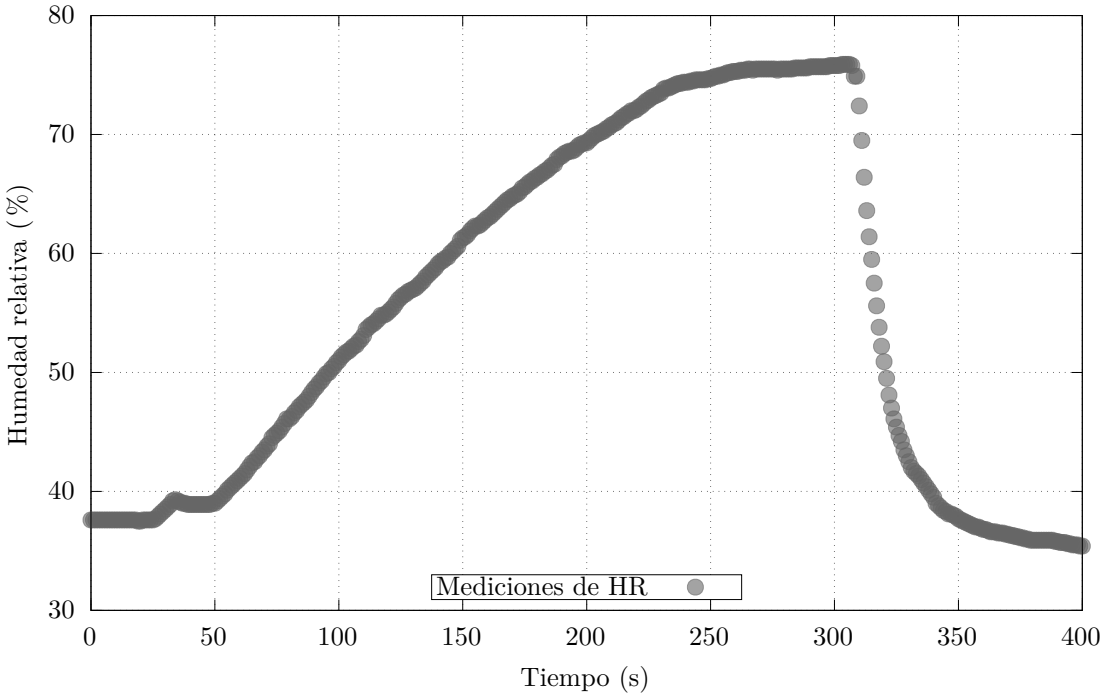


Figura 4.13: Gráfica de las mediciones de humedad relativa dentro de la atmósfera durante la "prueba no. 1" de adquisición del proceso de combustión.

En la Figura 4.13 se muestra la gráfica de la humedad relativa (en porcentaje) dentro de la atmósfera aislada durante la prueba de combustión. En esta gráfica se observa como la humedad comienza en un valor por debajo de 40 %. Este nivel aumenta levemente al introducirse la vela encendida (aproximadamente en el segundo 25), y después, una vez tapada la atmósfera (cerca del segundo 50), comienza a incrementar de forma casi constante. Este incremento comienza a desacelerar a medida que la llama de la vela se va reduciendo, hasta que finalmente se extingue y se estabiliza en el valor por debajo del 80 % (cerca del segundo 300). Después de esto, el valor de las mediciones de humedad cae rápidamente cuando atmósfera es destapada y ventilada, hasta llegar a un nivel inferior al inicial al final de la prueba.

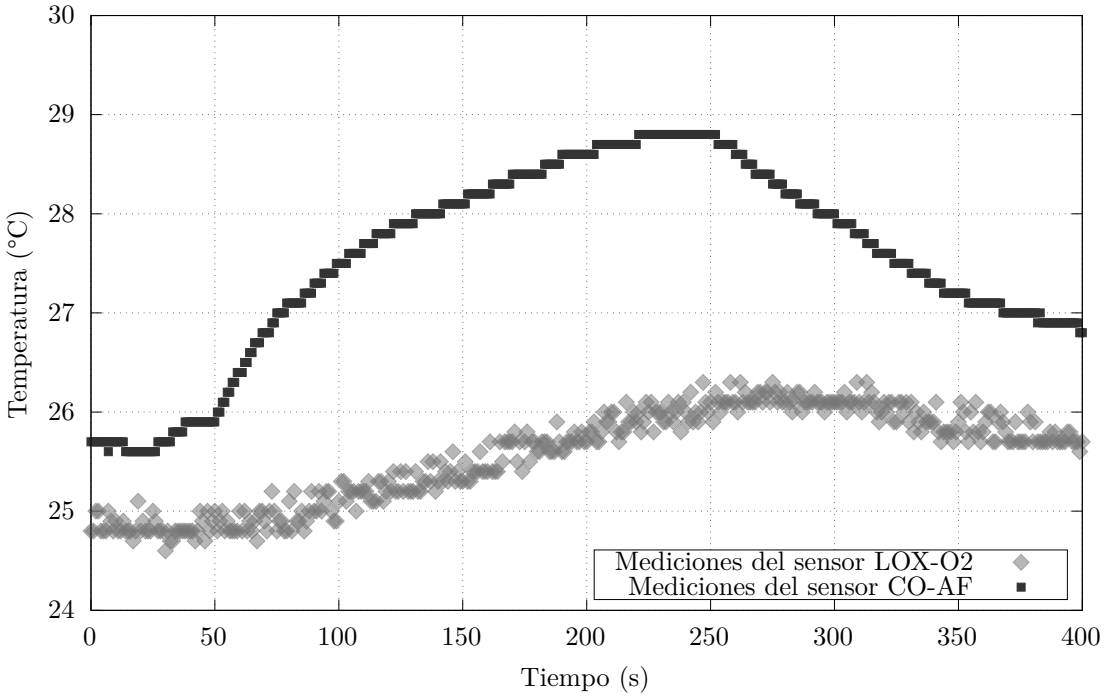


Figura 4.14: Gráfica de las mediciones de temperatura dentro de la atmósfera durante la "prueba no. 1" de adquisición del proceso de combustión.

En la Figura 4.14 se muestra la gráfica de las temperaturas medidas por los sensores *LOX-02*, de oxígeno, y *CO-AF*, de monóxido de carbono. En esta gráfica se puede observar que las mediciones de temperatura tiene un valor inicial de casi 25 °C en el sensor *LOX-02* y por debajo de 26 °C en el *CO-AF*. Después, ambas mediciones disminuyen levemente en el momento en el que se enciende la vela (aproximadamente en el segundo 25) y, una vez encendida, los valores de las mediciones del sensor *CO-AF* comienzan a aumentar lentamente hasta estabilizarse cerca de 26 °C antes del segundo 50. Después de esto, cuando la llama de la vela se estabiliza y a atmósfera se cierra, las mediciones del sensor *CO-AF* comienzan a incrementar notoriamente, mientras que las del *LOX-02* lo hacen más lentamente. El aumento de la temperatura medida por el sensor *CO-AF* comienza a desacelerar a medida que la llama de la vela se va extinguiendo hasta estabilizarse en casi 29 °C al apagarse completamente la vela (segundo 250 aproximadamente). Luego de esto, los valores de las mediciones comienzan a caer al destaparse y ventilarse la atmósfera. Por su parte, las mediciones del sensor *LOX-02* llegan a valores un poco mayores a 26 °C y estos comienzan a decender tiempo después de que la atmósfera es abierta.

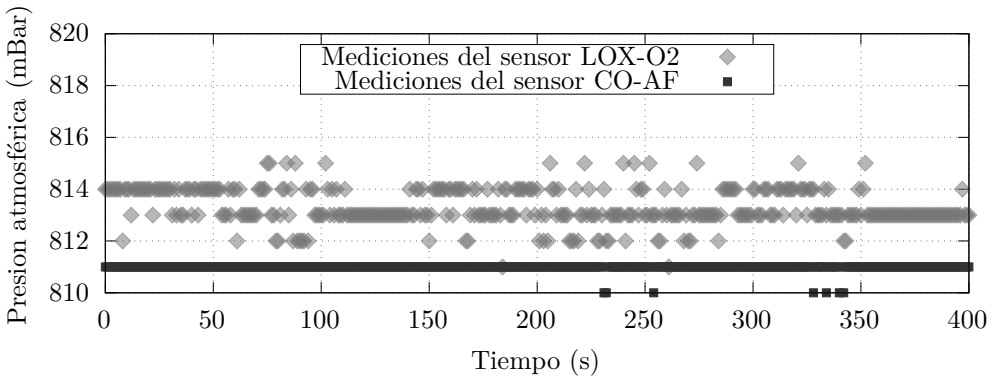


Figura 4.15: Gráfica de las mediciones de presión atmosférica dentro de la atmósfera durante la "prueba no. 1" de adquisición del proceso de combustión.

En la Figura 4.15 se muestran las mediciones de presión tomadas por los sensores *CO-AF* y *LOX-02* durante la prueba de combustión. Cómo se puede observar en esta gráfica la presión no varía considerablemente durante el proceso de combustión y las mediciones del sensor *CO-AF* se mantienen estables en poco más de 810 mBar y las del *LOX-02* al rededor de un valor inferior a 815 mBar. En esta gráfica también se puede observar que la estabilidad del sensor *CO-AF* es mayor que la del sensor *LOX-02*.

El comportamiento de estas variables en las repeticiones de esta prueba se mantienen similares. Las gráficas de las demás pruebas son presentadas en la Sección A.3 de esta tesis.

4.3.2 Comparación de los resultados obtenidos las prueba de adquisición de datos de la combustión

Los resultados de las repeticiones de la prueba de adquisición de datos de la combustión son comparados de la misma forma, observando los valores mínimos y máximos obtenidos, para analizar la fiabilidad del sistema de adquisición de datos desarrollado. Las gráficas de las variables medidas a través de las repeticiones de esta prueba, las cuales muestran sus comportamientos y valores mínimos y máximos, pueden ser revisadas en la Sección A.3 de este trabajo.

A continuación se presenta en la Tabla 4.3 la comparativa de las cinco pruebas de la adquisición en el proceso de combustión. En esta tabla se muestran las variables medidas en primera columna, seguidas de los indicadores de valores mínimos y máximos obtenidos en cada repetición de la prueba. Estos valores se muestran en las siguientes columnas, indicando su número de prueba, y finalmente, en la última columna se muestra la media calculada de los valores mínimos y máximos de las cinco repeticiones.

Para comparar los valores de las concentraciones de gases CO y CO_2 se emplean las mediciones con filtro, ya que estas fueron más estables, y para la temperatura y presión se utilizan las mediciones obtenidas por el sensor *CO-AF*, por su mayor estabilidad y velocidad de respuesta en en la temperatura, con respecto al sensor *LOX-02*.

Tabla 4.3: Tabla comparativa de valores mínimos y máximos de las variables medidas en varias repeticiones de la prueba de adquisición en el proceso de combustión.

		Pruebas de combustión					
Variables medidas		No. 1	No. 2	No. 3	No. 4	No. 5	Media
Concentración	Mín.	0	0	0	0	0	0.0
de ***CO*** (ppm)	Máx.	51	71	64	72	63	64.2
Concentración	Mín.	2,420	2,230	2,300	2,340	2,360	2,330
de $\boldsymbol{CO_2}$ (ppm)	Máx.	30,160	32,150	31,660	27,930	28,250	24,588
Concentración	Mín.	167,300	162,400	163,300	168,100	167,800	165,780
de $\boldsymbol{O_2}$ (ppm)	Máx.	203,100	204,400	203,900	203,900	203,800	203,820
Humedad	Mín.	37.5	34.2	35.2	36.3	36.0	35.84
relativa (%)	Máx.	73.4	76.6	76.1	70.5	69.3	73.18
Temperatura (°C)	Mín.	25.6	25.8	25.7	24.9	25.3	25.46
	Máx.	28.8	28.6	28.2	28.8	29.9	28.86
Presión (mBar)	Mín.	812	811	811	811	811	811.2
	Máx.	816	816	817	816	816	816.2

Una mejor visualización de la variación de los valores mínimos y máximos obtenidos en cada repetición puede visualizarse en la Tabla 4.4. En esta tabla se muestran las mayores variaciones (inferiores y superiores) de los valores mínimos y máximos obtenidos en las repeticiones de esta prueba, así como su valor relativo, tomando en cuenta la media como referencia. También se menciona en cuáles pruebas se dieron estas variaciones.

Tabla 4.4: Tabla de variaciones con respecto a la media de los valores mínimos y máximos obtenidos de las pruebas de adquisición del proceso de combustión.

		Variaciones			
Variables medidas		Desde	En la prueba	Hasta	En la prueba
Concentración	Mín.	-	-	-	-
de ***CO*** (ppm)	Máx.	-13.2 (-20.56 %)	No. 1	7.8 (12.15 %)	No. 4
Concentración	Mín.	-100 (-4.29 %)	No. 2	90 (3.86 %)	No. 1
de $\boldsymbol{CO_2}$ (ppm)	Máx.	-2,100 (-6.99 %)	No. 4	2,120 (7.06 %)	No. 2
Concentración	Mín.	-3,380 (-2.04 %)	No. 2	2,320 (1.40 %)	No. 4
de $\boldsymbol{O_2}$ (ppm)	Máx.	-720 (-0.35 %)	No. 1	580 (0.28 %)	No. 2
Humedad	Mín.	-0.6 (-2.20 %)	No. 4	0.3 (1.34 %)	No. 2
relativa (%)	Máx.	-0.7 (-2.29 %)	No. 3	1.0 (3.60 %)	No. 5
Temperatura (°C)	Mín.	-1.6 (-4.58 %)	No. 2	1.7 (4.63 %)	No. 1
	Máx.	-3.9 (-5.30 %)	No. 5	3.4 (4.67 %)	No. 2
Presión (mBar)	Mín.	-	-	-	-
	Máx.	-0.2 (-0.02 %)	No. 1, 2, 4 y 5	0.8 (0.10 %)	No. 3

Como puede observarse en la Tabla 4.4, la mayor variación de estos valores fue en los máximos de CO con 13.2 ppm por debajo de la media en la "prueba no. 1", los cuales representan un 20.56 % de la media los valores máximos, y 7.8 ppm por encima de la media en la "prueba no. 4", que equivalen al 12.15 % de esta media. Las demás variaciones se encuentran por debajo del 8 %, habiendo algunas muy bajas y otras donde no hubo variación, las cuales están representadas con un "-" en la tabla.

4.4 Conclusiones del capítulo

Las pruebas realizadas logran demostrar el correcto funcionamiento del sistema de adquisición y el programa desarrollado. Sin embargo, antes de haber podido realizar estas pruebas se tuvieron que resolver varios problemas que se fueron presentando.

Las pruebas realizadas de exhalación y combustión permitieron ver que las mediciones obtenidas tienen concordancia con los datos esperados. En las pruebas el oxígeno tiene un valor iniciar (superior a 20 %), cercano al la concentración de oxígeno en el aire (21 %), y el CO_2 en tiene un valor (superior a 2,000 ppm) cercano a las concentraciones esperadas en un aire poco contaminado(3,000 ppm). En la prueba de combustión el valor de oxígeno desciende a un poco más de 16 %, cercano al valor en el que casi ningún material arde (14 %). En la prueba de respiración las mediciones de CO_2 llegan hasta casi 40,000 ppm, que es la cantidad de bióxido de carbono en la exhalación de una persona. El comportamiento y el valor de las variables corresponde con los parámetros reportados en la literatura.

En la prueba de exhalación las variables con el comportamiento más notable fueron la concentración de CO_2 y oxígeno, estas se comportan como se esperaba, ya que el oxígeno cae al rededor de un 4 %, que es lo que aumenta de CO_2, por el proceso de respiración. Por otra parte, la humedad y la temperatura se incrementan con el vapor en la respiración y la temperatura corporal. Se puede observar que el comportamiento general en esta prueba se asemeja a un escalón amortiguado por la forma en que se

introducen los gases y luego se expulsan.

En la prueba de combustión las variables con el comportamiento más notable fueron el CO_2, el CO y el oxígeno. Durante la combustión, la concentración de oxígeno descendía, mientras que la de CO_2 y CO ascendían. A medida que la llama se iba extinguiendo, por la falta de oxígeno, la producción de CO_2 se iba frenando así como el decrecimiento de oxígeno. Por su lado, el CO se incrementa constantemente durante la combustión hasta extinguirse la llama, donde la concentración obtenía un impulso con el humo generado, aunque sin llegar a concentraciones altas debido en parte al gran volumen de la atmósfera asilada.

En los valores mínimos y máximos obtenidos no hubo gran variación de prueba en prueba. La mayor variación relativa fue la concentración de CO en la prueba de combustión (20.56 %), y esto puede deberse a que las concentraciones eran muy pequeñas, por lo que cualquier variación concentración de CO a esta escala se hace mayor. La segunda mayor variación relativa fue la concentración de CO_2 en la prueba de exhalación (8.42 %), y estas variaciones pueden estar ligadas al nivel de oxigenación en la sangre de la persona al momento de la prueba, o si estaba realizando alguna actividad física, etc. Las variaciones en las demás variables se encuentran por debajo del 8 %, con lo que se muestra la fiabilidad del sistema.

Capítulo 5

Conclusiones

Con la realización de este proyecto se logra el desarrollo de un sistema para la adquisición de concentraciones de gases y condiciones atmosféricas (presión, temperatura, humedad relativa), con el cuál se pretende asistir en la obtención de estas variables para el estudio de las reacciones sólido-gas en los tratamientos térmicos del acero. de forma que se cumple el objetivo principal de esta tesis y también se comprueba la hipótesis que plantea el aporte de las herramientas "*open source*" ya desarrolladas para aumentar las funciones y capacidades del sistema.

Dentro del desarrollo de este sistema también se implementó de un circuito de protección para señales digitales, el desarrollo de una librería para configuración y utilización de los puertos UART de la *Beaglebone* y un programa con funciones de configuración de sensores, obtención y registro de mediciones y gestión del tiempo para la adquisición, así como el diseño de una atmósfera parcialmente aislada. Trabajo y conocimiento que podrá ser utilizado en el futuro dentro de algún proyecto que requiera de estas herramientas.

La utilización de tarjetas controladoras facilitó la interfaz con los sensores, utilizando los puertos UART de la *Beaglebone* para comunicarse con éstos, disminuyendo así el posible ruido al transmitir señales analógicas. Estas tarjetas controladoras realizaron también el acondicionamiento y la compensación de las mediciones, disminuyendo así la afectación por el ruido y las condiciones atmosféricas. Para ello, las tarjetas controladoras utilizan sensores de temperatura, presión y humedad, que también pueden realizar mediciones para ser registradas por el sistema de adquisición.

Este sistema desarrollado puede medir concentraciones de hasta 250,000 ppm (25 %) de O_2, 5,000 ppm (0.5 %) de CO, y 200,000 ppm (20 %) de CO_2, en atmósferas con hasta 90 % de humedad relativa (sin condensación), con temperaturas desde 0 a 50 °C y una presión dentro del rango de 800 hasta 1,200 mbar. Estos rangos mencionados son para garantizar el uso de los tres sensores dentro de sus parámetros de operación.

Los resultados de las pruebas demuestran que tanto el algoritmo, los sensores y la interfaz de *hardware* (utilizando aisladores digitales ISOW7248) funcionan de manera adecuada individualmente, logrando así un correcto funcionamiento del sistema en conjunto. Estos resultados también demuestran la confiabilidad del sistema desarrollado, obteniendo un comportamiento consistente de las variables

medidas en los dos tipos de pruebas realizadas. También se observan pocas variaciones en las mediciones obtenidas en cada prueba, siendo del 8.42 % (con respecto a la media de los valores máximos de CO_2 obtenidos en las cinco diferentes pruebas) la mayor variación, en la prueba no. 2 de exhalación, y del 20.56 % en el nivel máximo de CO en la prueba no. 1 de combustión, posiblemente debido a las bajas concentraciones medidas.

El trabajo a futuro que se puede realizar en este sistema a corto y mediano plazo para lograr mejorarlo podría ser implementar funciones que alerten al usuario cuando los parámetros atmosféricas salgan del rango de operación de los sensores, así como mejorar la implementación de la duración de la adquisición de datos para introducir el tiempo exacto como texto (e. g.1:30:07, hh:mm:ss) y no sólo la cantidad de horas. También, opcionalmente para aumentar la modularidad del *software*, se podría separar la función `sensConf` en dos funciones, una para inicializar el sensor y otra para elegir modos de operación y otras opciones del sensor. En cuanto a los componentes de *hardware*, se podría diseñar una *daughterboard* con los aisladores digitales que se coloque encima de la *Beaglebone*, así como un módulo en el cuál se conecten todos los sensores de concentración de gases y también se conecten todos los cables de UART y alimentación en un sólo conector. La atmósfera controlada también podría mejorarse utilizando una tarjeta de control, en lugar de una fuente de alimentación para alimentar el mezclador de gases.

A largo plazo, el trabajo a futuro que se puede realizar en este sistema podría ser el desarrollo de aplicación web corriendo desde la *Beaglebone*, para un acceso remoto desde un navegador web, con una interfaz gráfica que facilite la utilización del sistema al usuario para la realización de adquisición de datos en pruebas y experimentos, la visualización y la obtención de gráficas y datos, y el monitoreo en tiempo real de las pruebas y experimentos.

Antes de obtener el manual del sensor EC200, no se conocían los comandos utilizados para su operación, por lo que el sensor tuvo que utilizarse con el programa *GasLab*, proporcionado por el provedor co2meter. Al comunicarse por medio de este programa, se monitorearon los comandos utilizados por éste y las respuestas del sensor, utilizando dos terminales RX de los puertos UART de la *Beaglebone* y el programa *minicom*, para conocer la forma de operación del sensor.

También se puede desarrollar una función para la gestión del tiempo de adquisición, utilizando los PRU de la *Beaglebone*, para así mejorar el control de la obtención de las mediciones y transladar esa carga de trabajo a un microcontrolador con capacidades en tiempo real.

Los puertos UART de la *Beaglebone* no se configuraban correctamente utilizando la estructura de `termios` en el programa de adquisición de datos. Esto era debido a que al modificar los atributos de los puertos no se limpiaban los valores previos, por lo que se mantenían algunas configuraciones no deseadas. Para corregir este problema sólo se limpiaron las configuraciones existentes en los puertos antes de asignar los nuevos parámetros.

La configuración de los sensores en algunas ocasiones no era exitosa, por lo que era necesario realizar más intentos. Este error se manejó recibiendo el mensaje de confirmación del sensor y utilizando un ciclo `while` para realizar un determinado número de intentos hasta que el sensor confirmara su correcta configuración.

También fue necesario examinar los mensajes de respuesta del sensor *SprintIR*, ya que este no respondía con el mensaje de confirmación de configuración especificado en el manual del fabricante. Para esto se creó una sección de código imprimía en la terminal los caractéres del mensaje recibido y su valor en código ASCII. Este código fue retirado de la versión actual del programa.

El sistema operativo utilizado en este trabajo no ejecuta las tareas en tiempo real, por lo que se quiso anticipar al problema de acumular un retardo en las mediciones durante pruebas muy prolongadas

al utilizar la función `sleep`. Al principio se utilizó un ciclo `while`, donde el programa se detenía hasta que la hora/fecha del sistema (representada en segundos) llegaba al momento en el que se realizaba el muestreo, pero al implementar este algoritmo se percató de que el CPU se sobrecargaba al estar constantemente verificando la hora/fecha del sistema. Debido a esto se decidió poner al programa en reposo el tiempo restante para el siguiente muestreo después de haber realizado y almacenado las mediciones (menos un margen de tiempo considerable para darle oportunidad al administrador de tareas regresar al programa antes de este muestreo), utilizando la función `sleep` en el algoritmo, esto ayudó a reducir el uso de CPU a un valor al rededor del 20 %.

Apéndice A

Anexo o Apéndice

A.1 Códigos

En esta sección del Apéndice A se presentan los archivos del código fuente utilizados para el desarrollo del programa de este trabajo. En el Código A.1 se muestra el archivo principal `main.c` con la función `main` del programa. En el Código A.2 se muestra el archivo de encabezado (archivo H) `daq.h` con las librerías, las definiciones, y la declaración de las funciones necesarias para la adquisición de datos. En el Código A.3 se muestra el archivo de implementación (archivo C) `daq.c`, donde se encuentran las funciones para la adquisición de datos. En el Código A.4 se muestra el archivo de encabezado (archivo H) `daq.h` con las librerías, las definiciones, y la declaración de las funciones y las variables necesarias para la comunicación a través de los puertos UART. Y por último, en el Código A.5 se muestra el archivo de implementación (archivo C) `daq.c`, donde se encuentran las funciones y la declaración de la estructura `termios` para la comunicación a través de los mismos puertos.

Código A.1: Archivo main.c

```
1  #include "daq.h"
2
3  int main(int argc, char *argv[]){
4
5      int T_acq, t_samp;
6
7      if(argc!=3){
8          printf("Invalid number of arguments,\n"\
9          "Usage: daq <daq_duration(h)> <sampling_period(s)>\nexiting!\n");
10         return -2;
11     }
12
13     // configuring sensors
14     printf("Configuring CO2 sensor\n");
15     sensConf(SprintIR, B9600, Polling_Mode, " K 00002\r\n", 5);
16     sensConf(SprintIR, B9600, FILnUNFIL, " M 00006\r\n", 5);
17     printf("Configuring CO sensor\n");
18     sensConf(COAF, B9600, Polling_Mode, "K 00002\r\n", 5);
19     sensConf(COAF, B9600, M_zZTHB, "M 14406\r\n", 5);
20     printf("Configuring O2 sensor\n");
21     sensConf(LOX_O2, B9600, OX_P_Mode, "M 01\r\n", 5);
22
23     // acquiring
```

```
    T_acq = atoi(argv[1]);//str to int
    t_samp = atoi(argv[2]);
    printf("Starting data acquisition with duration of %dh every %ds\n", T_acq, t_samp);
    DAQ(T_acq, t_samp);

    // deinit uart
    uartClose(SprintIR);
    uartClose(COAF);
    uartClose(LOX_O2);
    // finishing
    printf("Exiting of the program...\n");
    return 0;
}
```

Código A.2: Archivo daq.h

```

#ifndef DAQ_H
#define DAQ_H

//HEADERS
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <stdint.h>
#include "uart.h"

//DEFINITIONS
#define SprintIR  4    //co2 sensor
#define COAF      1    //co sensor
#define LOX_O2    2    //o2

//comands
#define Rep_Dev_ID      "Y\r\n"
#define Polling_Mode    "K_2\r\n"
#define OX_P_Mode       "M_1\r\n"
#define FILnUNFIL       "M_6\r\n"
#define unf_gas_con     "z\r\n"
#define fil_gas_con     "Z\r\n"
#define Temperature     "T\r\n"
#define get_readigns    "Q\r\n"
#define percent_oxigen  "%\r\n"
#define ppm_oxigen      "O\r\n"
#define M_zZTHBD        "M_14406\r\n"
#define M_zZTHB         "M_12358\r\n"
#define Readings_OX     "A\r\n"

//FUNCTIONS
int sensConf(uint8_t uartNumber, int baudRate, char mode[], char response[], int tries);
int DAQ(int t_hrs, int sp_s);
char *getMeasures(char src[], char fval, int nchar);

#endif
```

Código A.3: Archivo daq.c

```

#include "daq.h"

//DEFINITIONS
#define data_file_path  "./DATA/data.dat"

int sensConf(uint8_t uartNumber, int baudRate, char mode[], char response[], int tries)
{
        int count;

        uartConf(uartNumber, baudRate);
        while(tries){
                uartTransmit(uartNumber, mode);
                tcdrain(uartFile[uartNumber]); //wait all data has been sent
                printf("Command_sended.\n");
                count = uartReceive(uartNumber);
```

```
                if (count == 0) printf("There was no data available to read!\n");
                else if (strcmp(receive[uartNumber], response) == 0) {
                        printf("Sensor configurated.\n");
                        return 0;
                } else {
                        printf("The following was read in [%d]: %s\n",count,receive[uartNumber]);
                }
                tries --;
        }
        printf("Sensor configuration failed.\n");
        return -1;
}

int DAQ(int t_hrs, int sp_s)
{
        FILE* dfp; // create a file pointer
        const char data_header[200] =
                "t\tCO2\tCO2_f\tCO\tCO_f\tO2\tO2\tT_CO\tT_O2\tP_CO\tP_O2\tRH\n"\
                "s\tppm\tppm\tppm\tppm\tppO2\t%\tC*10\tC\tmBar\tmBar\t%";
        //measurements varirables
        char co2_uf[10]="",     co2_f[10]="",
                co_uf[10]="",   co_f[10]="",
                o2_ppm[10]="",  o2_xcent[10]="",
                co_temp[10]="", o2_temp[10]="",
                co_press[10]="",o2_press[10]="",
                co_relH[10]="", DATA[200]="";
        time_t curtime; //current time (date)
        clock_t start_t, end_t; //processing time measurements variables
        time_t next_samp_time, t0; //time control variables
        //maximum cycle iteration time
        double iteration_time_ms = sp_s*1e6 - 0.2e6;
        double inactivity_time = 1; //time to sleep

        time(&curtime); //saving date in curtime
        //write start data
        dfp = fopen(data_file_path, "w"); // open file for writing
        // send the value to the file
        fprintf(dfp, "Starting DAQ at %s\n", ctime(&curtime));
        fclose(dfp); // close the file using the file pointer
        printf("%s", colums); //display variables colums

        next_samp_time = time(NULL)+1;//setting next sampling time
        t0 = next_samp_time;//saving initial time
        //cycle from 0 to adquisition time (seconds), incrementing sampling period
        for(time_t t = 0; t < (t_hrs*3600); t+=sp_s){

        if (inactivity_time <= 1){ //checking inactivity time
                usleep((int)inactivity_time); //inactivity
        }
        else{
                printf("Ejecution_time_exceded.\n%s", colums);
                next_samp_time = time(NULL)+1; //refreshing time values
                t = next_samp_time-t0;
        }
        //synchronizing/waiting to start measurements
        while(next_samp_time != time(NULL));

        start_t = clock(); //saving start time
          //transmiting commands to sensors
          uartTransmit(COAF, get_readigns);
          uartTransmit(LOX_O2, Readings_OX);
          uartTransmit(SprintIR, get_readigns);
          uartReceive(COAF); //receiving replys from
          uartReceive(LOX_O2);
          uartReceive(SprintIR);
          //interpreting and spliting measurements in variables
          memcpy(co2_uf,  getMeasures(receive[SprintIR],'z', 5), 5);
          memcpy(co2_f,   getMeasures(receive[SprintIR],'Z', 5), 5);
          memcpy(co_uf,   getMeasures(receive[COAF],    'z', 5), 5);
          memcpy(co_f,    getMeasures(receive[COAF],    'Z', 5), 5);
          memcpy(o2_ppm,  getMeasures(receive[LOX_O2],  'O', 6), 6);
          memcpy(o2_xcent,getMeasures(receive[LOX_O2],  '%', 6), 6);
          memcpy(co_temp, getMeasures(receive[COAF],    'T', 5), 5);
          memcpy(o2_temp, getMeasures(receive[LOX_O2],  'T', 5), 5);
          memcpy(co_press,getMeasures(receive[COAF],    'B', 5), 5);
          memcpy(o2_press,getMeasures(receive[LOX_O2],  'P', 4), 4);
          memcpy(co_relH, getMeasures(receive[COAF],    'H', 5), 5);
```

```
94            //saving formated measurements in string DATA
95            sprintf(DATA,
96                    "%d\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s",
97                    (int)t, co2_uf, co2_f, co_uf, co_f, o2_ppm, o2_xcent,
98                    co_temp, o2_temp, co_press, o2_press, co_relH);
99            printf("\r%s", DATA); //showing measurements on display
100
101           dfp = fopen(data_file_path, "a"); // open file for writing
102           // saving measurements string to data file
103           fprintf(dfp, "%s\n", DATA);
104           fclose(dfp); // close the file using the file pointer
105           next_samp_time += sp_s; //set next sampling time
106         end_t = clock(); //saving end time
107         //calculate time to sleep
108         inactivity_time = iteration_time_ms - ((double)(end_t - start_t)*1e6 / CLOCKS_PER_SEC);
109         }
110
111         return 0;
112 }
113
114 char *getMeasures(char src[], char fval, int nchar)
115 {
116         char * ptr = &src[0];
117         static char s[10]="";
118
119         while(*ptr != fval) ptr++;
120         ptr += 2;
121         memcpy(s, ptr, nchar);
122         return s;
123 }
```

Código A.4: Archivo uart.h

```
1
2  #ifndef UART_H
3  #define UART_H
4
5  // Includes begin
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <stddef.h>
9  #include <time.h>
10 #include <termios.h>
11 #include <fcntl.h>
12 #include <unistd.h>
13 #include <sys/types.h>
14 #include <string.h>
15 #include <stdint.h>
16
17 // Error codes and return values
18 #define UART_FUNCTION_SUCCESSFUL        0
19 #define UART_NUMBER_INCORRECT           1
20
21 //VARIABLES
22 int uartFile[6];           //file descriptor
23 char receive[6][100];      //declare a buffer for receiving data
24
25 // Function declarations
26 int uartConf(uint8_t uartNumber, int baudRate);
27 int uartClose(uint8_t uartNumber);
28 int uartTransmit(uint8_t uartNumber, char message[]);
29 int uartReceive(uint8_t uartNumber);
30
31 #endif // UART_H
```

Código A.5: Archivo uart.c

```
1
2  #include "uart.h"
3
4  //file paths and global variables
5  #define incomplet_uart_path "/dev/ttyO"
6  struct termios options;            //The termios structure
7
8  //    *** FUNCTIONS ***
```

```c

int uartConf(uint8_t uartNumber, int baudRate)
{
        char fullFileName[11];

        //completing uart file path
        if ((uartNumber==1)||(uartNumber==2)||(uartNumber==4)||(uartNumber==5))
                sprintf(fullFileName, incomplet_uart_path "%d", uartNumber);
        else{
                perror("Wrong UART number. " \
                "UART numbers available 1, 2, 4 or 5.\n");
                return UART_NUMBER_INCORRECT;
        }

        //openign uart file
        printf("Configuring UART%d.\n", uartNumber);
        if ((uartFile[uartNumber] = open(fullFileName, O_RDWR | O_NOCTTY | O_NDELAY | O_NONBLOCK))<0){
                perror("UART: Failed to open the file.\n");
                return -1;
        }

        //Sets the parameters associated with file
        tcgetattr(uartFile[uartNumber], &options);
        //cleaning flags
        options.c_ispeed = 0;   options.c_lflag = 0;
        options.c_line = 0;     options.c_oflag = 0;
        options.c_ospeed = 0
        // Set up the communications options:
        options.c_cflag = baudRate | CS8 | CREAD | CLOCAL; // 8-bit, enable receiver, no modem control lines
        options.c_iflag = IGNPAR; //ignore partity errors, CR -> newline
        tcflush(uartFile[uartNumber], TCIOFLUSH); //discard file information not transmitted
        tcsetattr(uartFile[uartNumber], TCSANOW, &options); //changes occur immmediately
        printf("UART%d configurated.\n", uartNumber);

        return UART_FUNCTION_SUCCESSFUL;
}

int uartClose(uint8_t uartNumber)
{
    printf("Closing UART%d.\n", uartNumber);
    close(uartFile[uartNumber]);
    return UART_FUNCTION_SUCCESSFUL;
}

int uartTransmit(uint8_t uartNumber, char message[])
{
   int count;

   //writing file
   if ((count = write(uartFile[uartNumber], message, (strlen(message))))<0){        //send the string
      perror("Failed to write to the output\n");
      return -1;
   }
   tcflush(uartFile[uartNumber], TCOFLUSH);

   return UART_FUNCTION_SUCCESSFUL;
}

int uartReceive(uint8_t uartNumber)
{
    int count;

    while((count = read(uartFile[uartNumber], (void*)receive[uartNumber], 100)) < 0);
    tcflush(uartFile[uartNumber], TCIFLUSH);

    return count;
}
```

A.2 Tablas de datos

En esta sección se muestran sólo de forma ilustrativa las mediciones obtenidas de la prueba de Medición de gases de exhalación no. 1 en la Tabla A.1.

Tabla A.1: Tabla de mediciones.

t. s	CO_2 ppm	CO_2 f. ppm	CO ppm	CO f. ppm	O_2 ppO_2	O_2 %	T. CO °C	T. O_2 °C	P. CO mbar	P. O_2 mbar	H. R. %
0	2,240	2,170	0	0	165.1	20.36	23.6	23.0	813	811	41.9
1	2,160	2,160	216	0	165.0	20.35	23.6	23.0	813	811	41.9
2	2,170	2,160	0	0	165.1	20.35	23.6	23.0	813	811	41.9
3	2,210	2,170	1	0	165.1	20.35	23.6	22.9	813	811	41.9
4	2,160	2,150	1	0	165.1	20.36	23.6	22.8	813	811	41.9
5	2,080	2,170	0	0	165.1	20.36	23.6	22.8	814	811	41.9
6	2,200	2,190	1	0	165.2	20.36	23.6	22.9	813	811	41.9
7	2,180	2,170	0	0	164.8	20.37	23.6	23.0	815	811	41.9
8	2,110	2,170	1	0	164.7	20.32	23.6	23.1	813	811	41.9
9	2,080	2,140	0	0	164.7	20.32	23.6	23.1	814	811	41.9
10	2,080	2,140	0	0	164.7	20.31	23.6	23.0	814	811	41.9
11	2,160	2,170	1	0	164.7	20.31	23.6	22.9	813	811	41.9
12	2,190	2,160	0	0	164.7	20.31	23.6	23.0	813	811	41.9
13	2,100	2,170	0	0	164.7	20.31	23.6	23.0	813	811	41.9
14	2,160	2,170	1	0	164.7	20.31	23.6	23.0	813	811	41.9
15	2,170	2,150	0	0	164.8	20.31	23.6	22.8	813	811	41.9
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
144	38,320	38,370	0	0	138.0	17.02	23.8	23.1	813	811	83.3
145	38,320	38,470	0	0	138.0	17.02	23.8	23.2	814	811	83.3
146	37,680	38,270	0	0	138.0	17.02	23.8	23.3	813	811	83.3
147	38,370	38,220	0	0	138.0	17.02	23.8	23.2	814	811	83.2
148	38,900	38,370	0	0	138.0	17.02	23.8	23.1	814	811	83.1
149	38,660	38,370	0	0	138.0	17.02	23.8	23.2	814	811	83.1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
160	38,560	38,370	0	0	137.9	17.01	23.8	23.3	814	811	82.7
161	38,800	38,320	0	0	137.9	17	23.8	23.1	814	811	82.7
162	38,370	38,270	0	0	137.9	17	23.8	23.0	814	811	82.6
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
493	2,190	2,250	0	0	164.9	20.33	23.6	22.9	813	811	42.2
494	2,180	2,230	1	0	164.9	20.33	23.6	23.0	813	811	42.2
495	2,190	2,240	0	0	165.3	20.33	23.6	22.9	813	811	42.2
496	2,160	2,230	1	0	165.3	20.38	23.6	23.0	813	811	42.2
497	2,240	2,260	1	0	165.2	20.38	23.6	23.0	813	811	42.3
498	2,300	2,260	0	0	164.7	20.37	23.6	23.2	813	811	42.3
499	2,190	2,240	1	0	164.6	20.31	23.6	22.9	813	811	42.3
500	2,280	2,230	0	0	164.6	20.31	23.6	22.9	813	811	42.3
501	2,170	2,240	0	0	164.7	20.3	23.6	23.0	813	811	42.3
502	2,140	2,230	0	1	164.7	20.3	23.6	23.0	813	811	42.3
503	2,300	2,250	0	0	164.7	20.3	23.6	23.0	813	811	42.3

A.3 Gráficas de los datos obtenidos de las pruebas

En esta sección se presentan las gráficas realizadas con los datos obtenidos de las pruebas de medición de gases de exhalación y adquisición de datos del proceso de combustión en su sección correspondiente con el número de prueba.

A.3.1 Medición de gases de exhalación: Prueba no. 2

En esta sección se muestran las graficas de las mediciones de CO en la Figura A.1, $CO2$ en la Figura A.2, O_2 en la Figura A.3, humedad relativa en la Figura A.4, temperatura en la Figura A.5 y presión en la, de la prueba de mediciones de gases de exhalación no. 2.

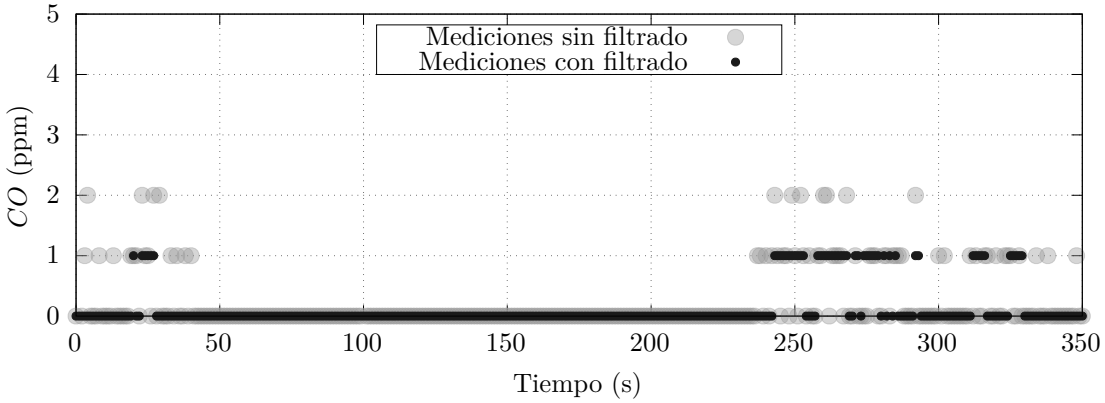


Figura A.1: Gráfica de las mediciones de CO durante la "prueba no. 2" de medición de gases de exhalación.

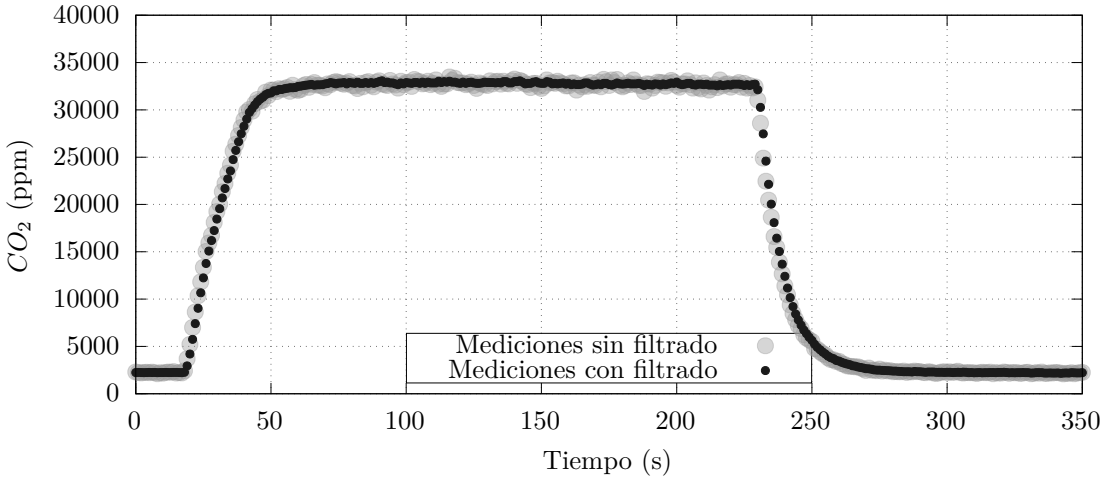


Figura A.2: Gráfica de las mediciones de CO_2 durante la "prueba no. 2" de medición de gases de exhalación.

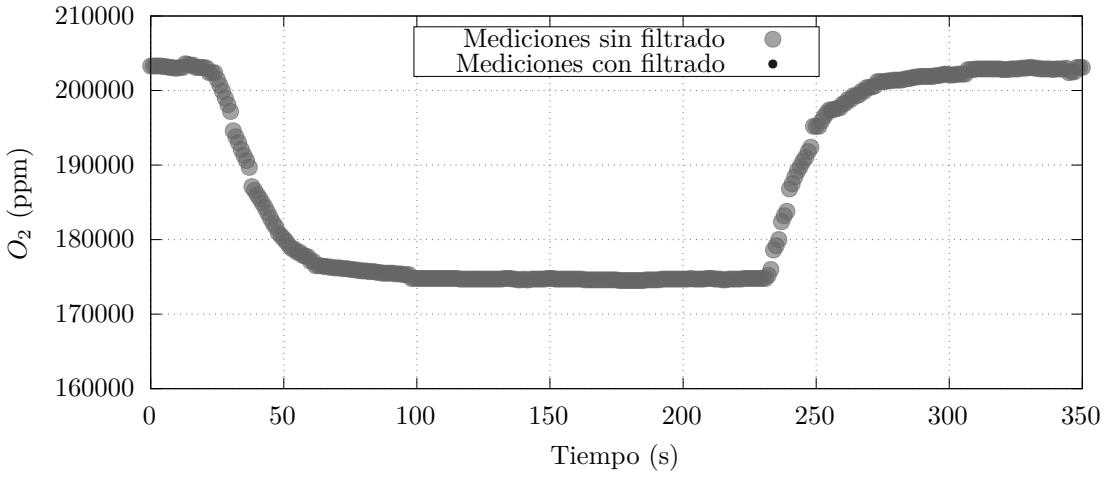


Figura A.3: Gráfica de las mediciones de O_2 durante la "prueba no. 2" de medición de gases de exhalación.

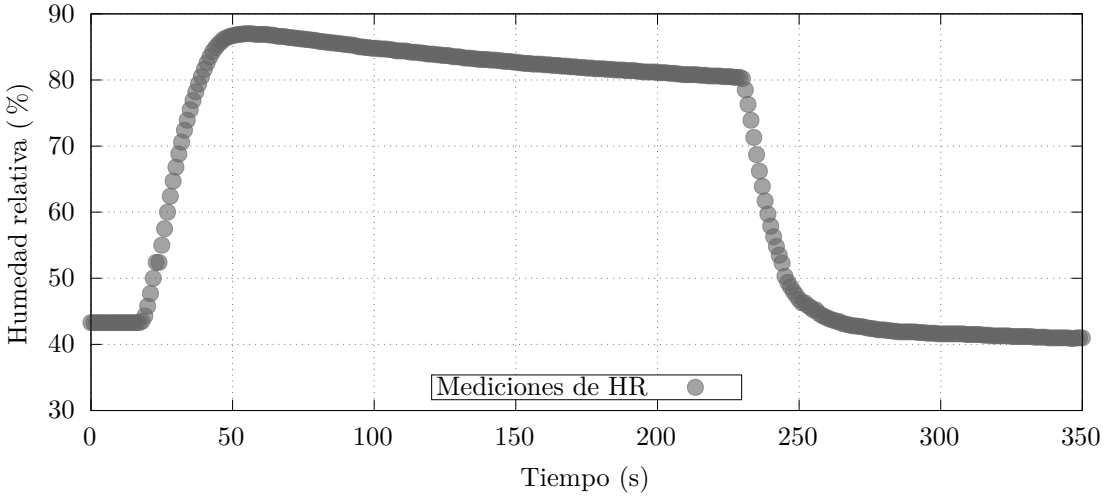


Figura A.4: Gráfica de las mediciones de humedad relativa durante la “prueba no. 2” de medición de gases de exhalación.

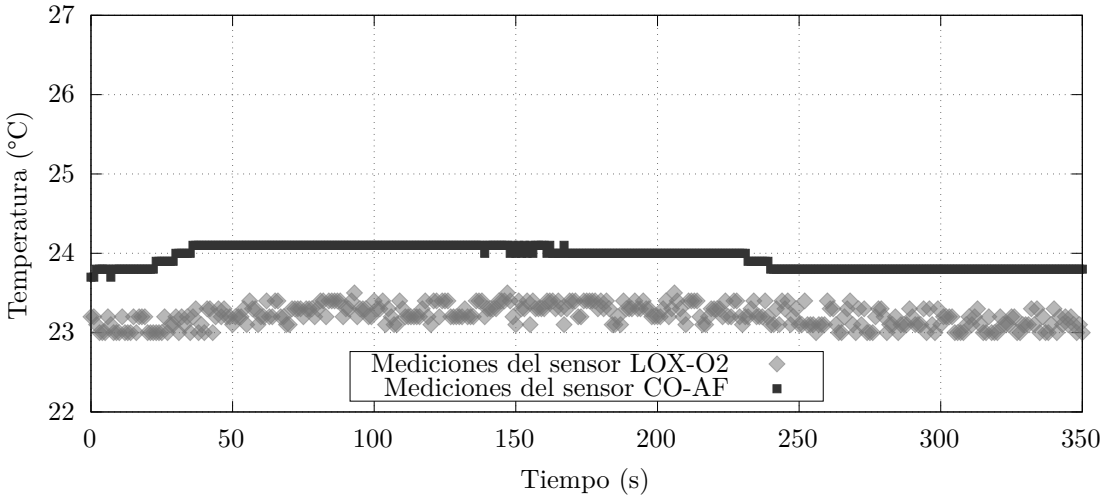


Figura A.5: Gráfica de las mediciones de temperatura durante la “prueba no. 2” de medición de gases de exhalación.

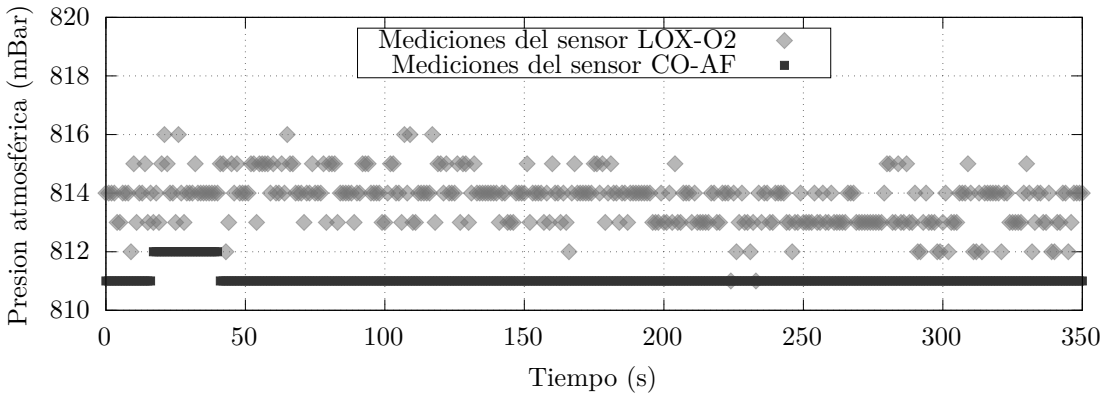


Figura A.6: Gráfica de las mediciones de presión atmosférica durante la “prueba no. 2” de medición de gases de exhalación.

A.3.2 Medición de gases de exhalación: Prueba no. 3

En esta sección se muestran las graficas de las mediciones de CO en la Figura A.7, $CO2$ en la Figura A.8, O_2 en la Figura A.9, humedad relativa en la Figura A.10, temperatura en la Figura A.11 y presión en la, de la prueba de mediciones de gases de exhalación no. 3.

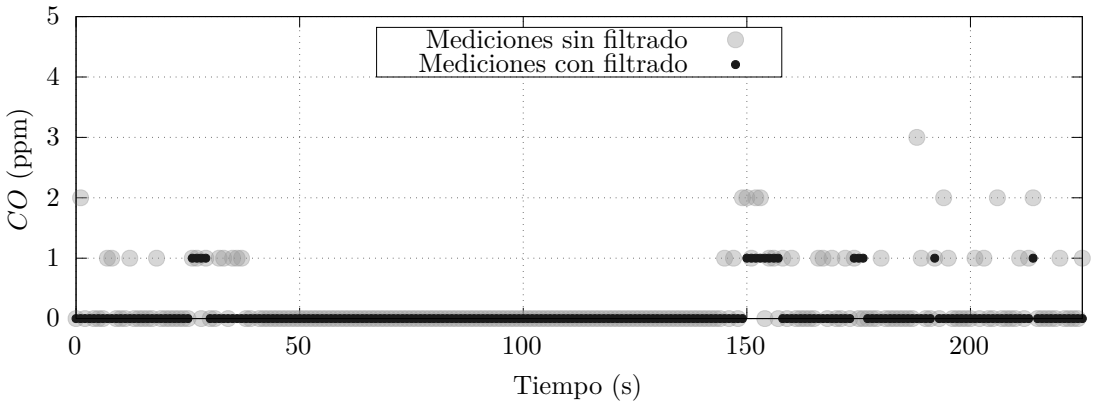


Figura A.7: Gráfica de las mediciones de CO durante la "prueba no. 3" de medición de gases de exhalación.

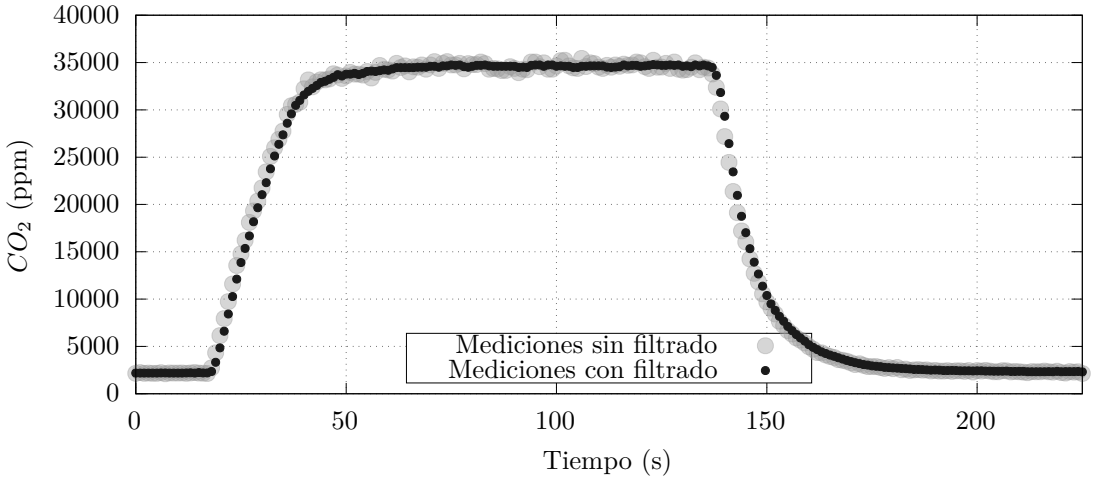


Figura A.8: Gráfica de las mediciones de CO_2 durante la "prueba no. 3" de medición de gases de exhalación.

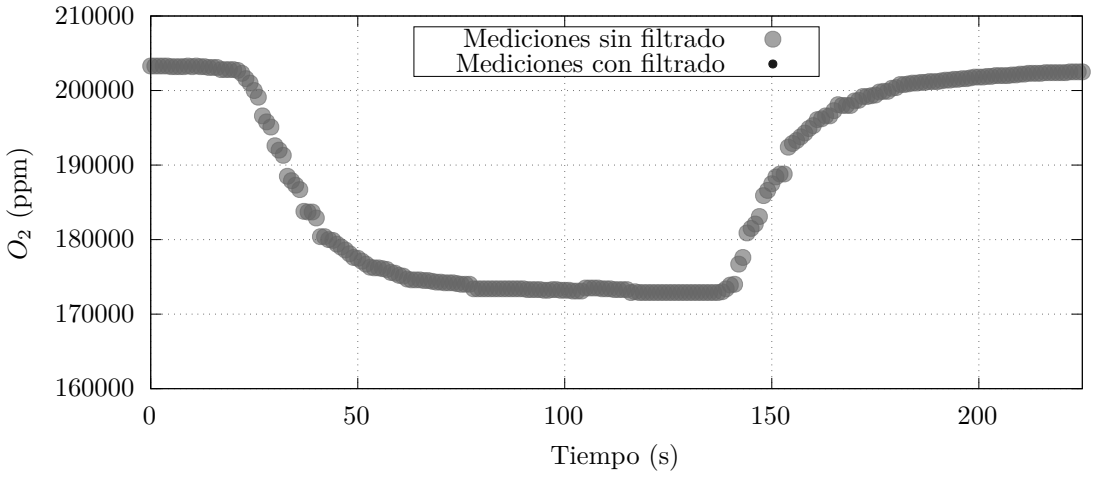


Figura A.9: Gráfica de las mediciones de O_2 durante la "prueba no. 3" de medición de gases de exhalación.

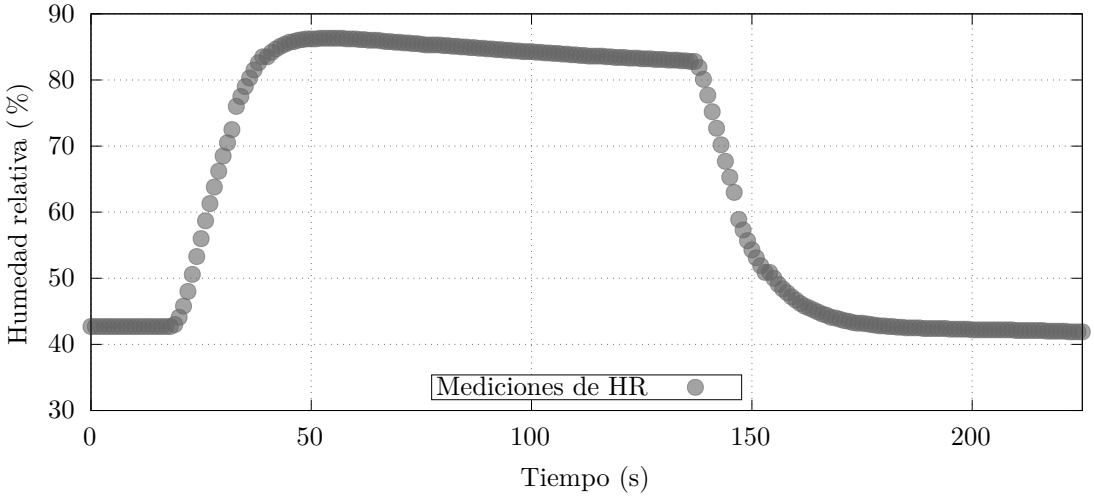


Figura A.10: Gráfica de las mediciones de humedad relativa durante la “prueba no. 3” de medición de gases de exhalación.

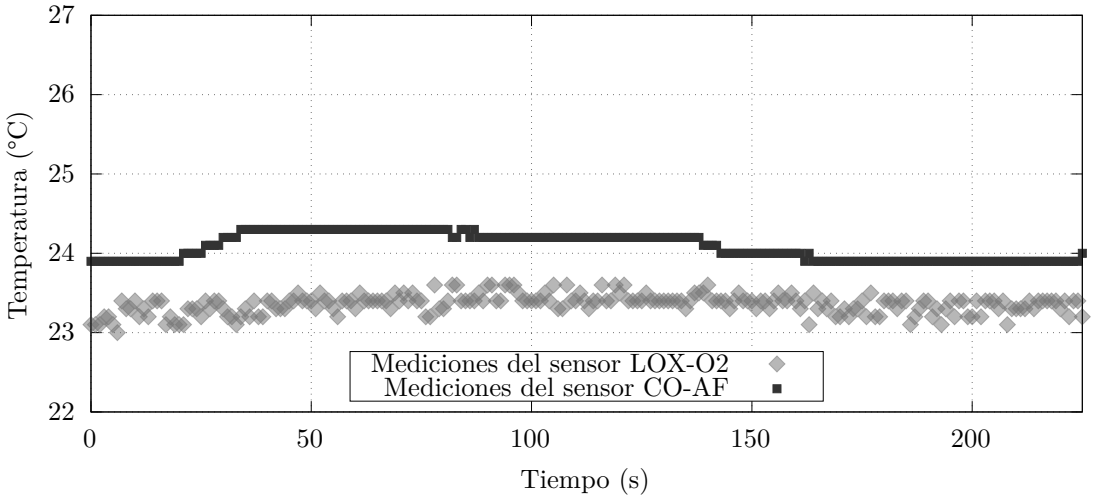


Figura A.11: Gráfica de las mediciones de temperatura durante la “prueba no. 3” de medición de gases de exhalación.

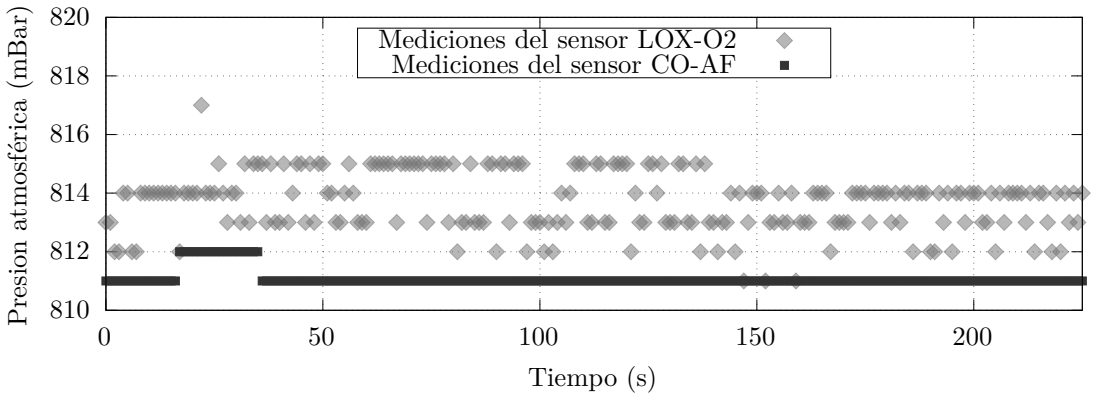


Figura A.12: Gráfica de las mediciones de presión atmosférica durante la “prueba no. 3” de medición de gases de exhalación.

A.3.3 Medición de gases de exhalación: Prueba no. 4

En esta sección se muestran las graficas de las mediciones de CO en la Figura A.13, $CO2$ en la Figura A.14, O_2 en la Figura A.15, humedad relativa en la Figura A.16, temperatura en la Figura A.17 y presión en la, de la prueba de mediciones de gases de exhalación no. 2.

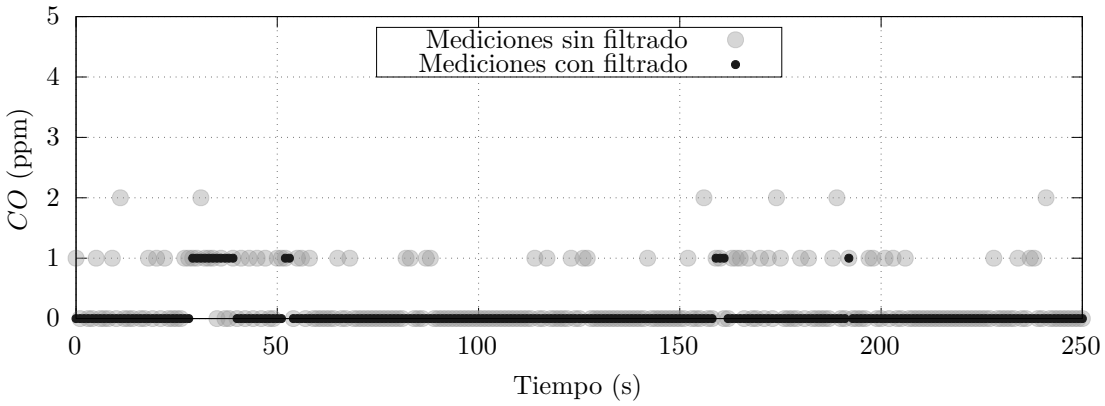


Figura A.13: Gráfica de las mediciones de CO durante la "prueba no. 4" de medición de gases de exhalación.

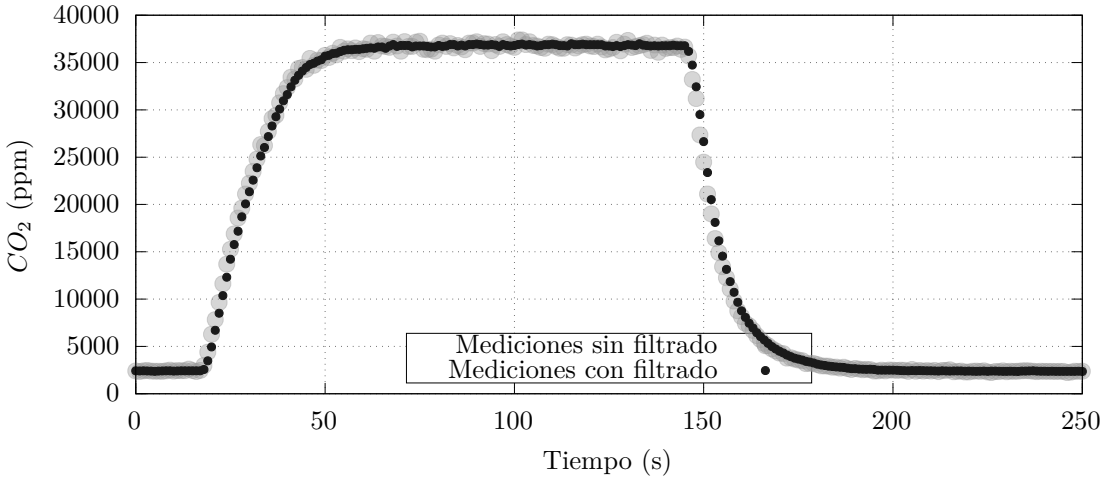


Figura A.14: Gráfica de las mediciones de CO_2 durante la "prueba no. 4" de medición de gases de exhalación.

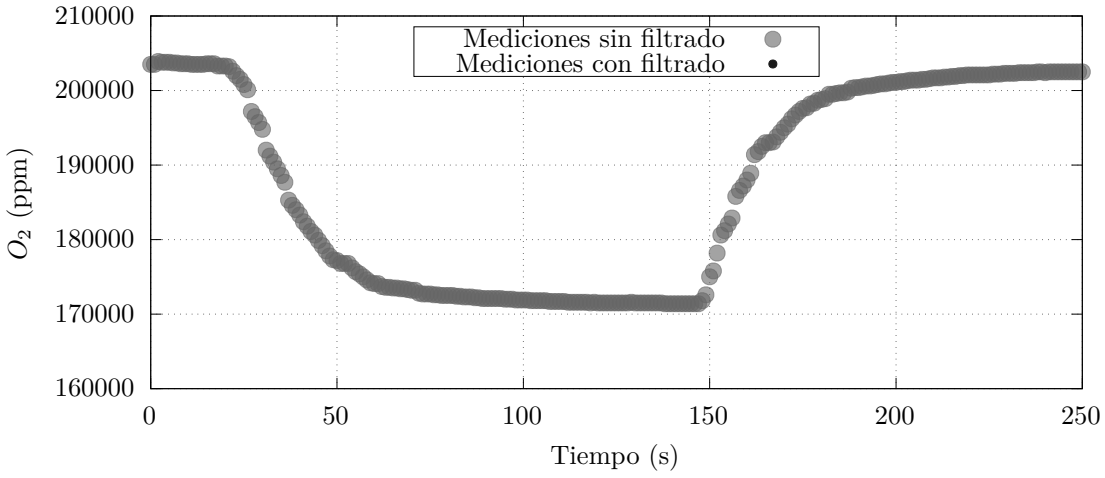


Figura A.15: Gráfica de las mediciones de O_2 durante la "prueba no. 4" de medición de gases de exhalación.

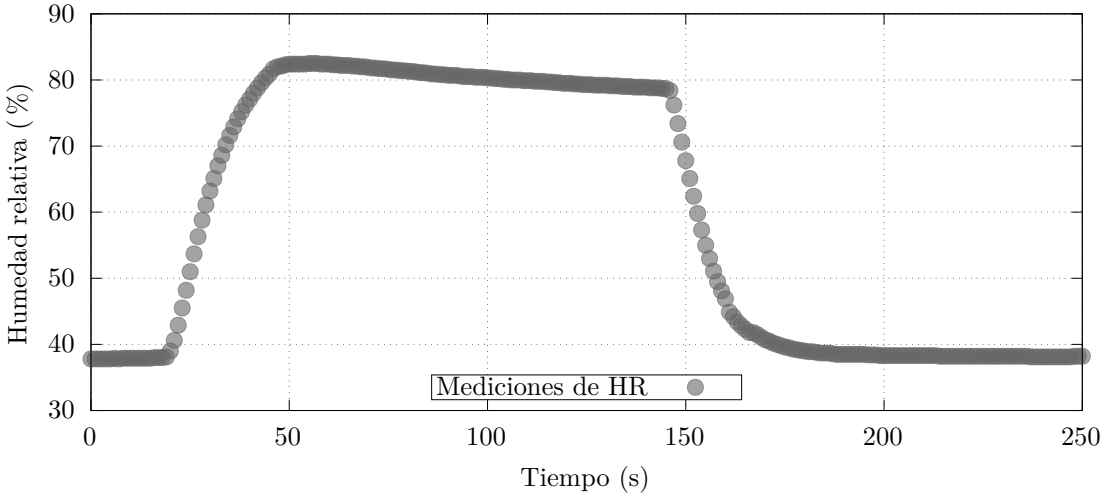


Figura A.16: Gráfica de las mediciones de humedad relativa durante la “prueba no. 4” de medición de gases de exhalación.

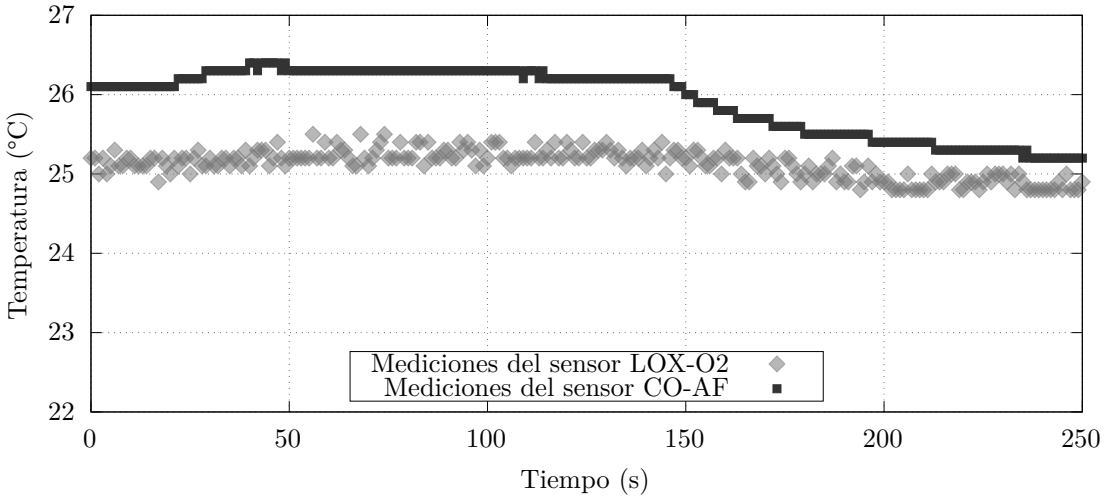


Figura A.17: Gráfica de las mediciones de temperatura durante la “prueba no. 4” de medición de gases de exhalación.

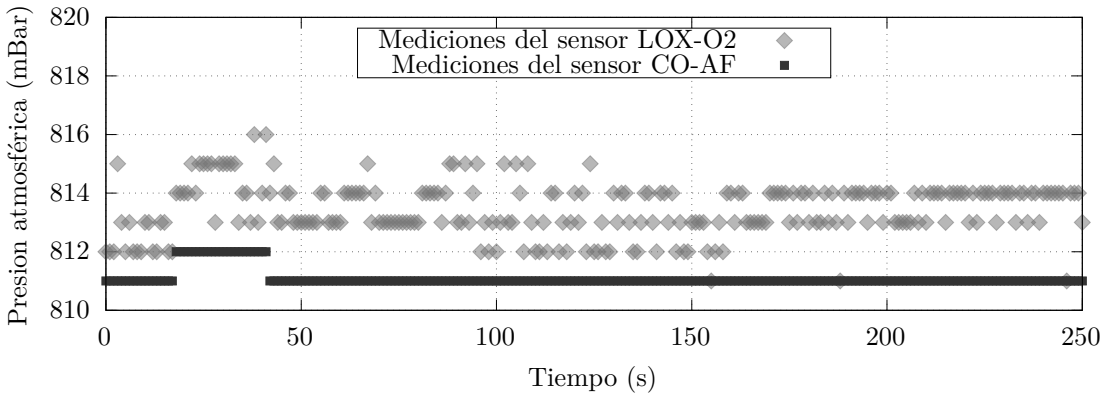


Figura A.18: Gráfica de las mediciones de presión atmosférica durante la “prueba no. 4” de medición de gases de exhalación.

A.3.4 Medición de gases de exhalación: Prueba no. 5

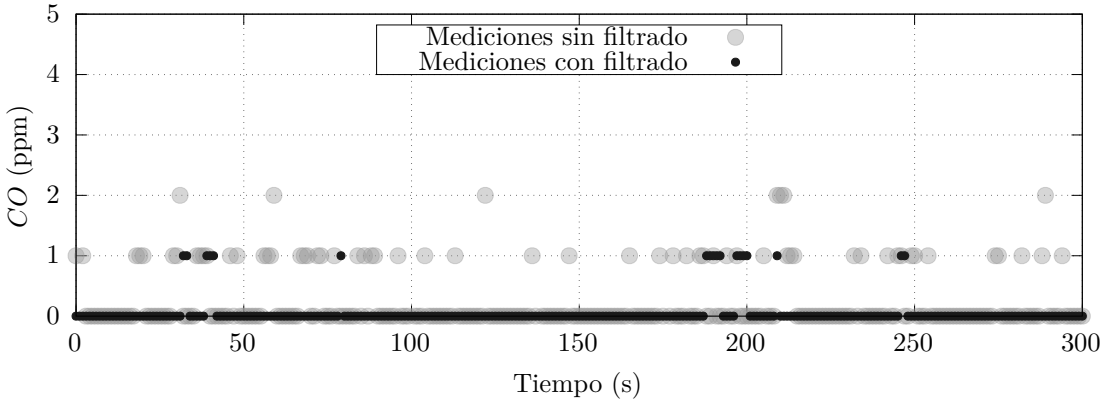


Figura A.19: Gráfica de las mediciones de CO durante la "prueba no. 5" de medición de gases de exhalación.

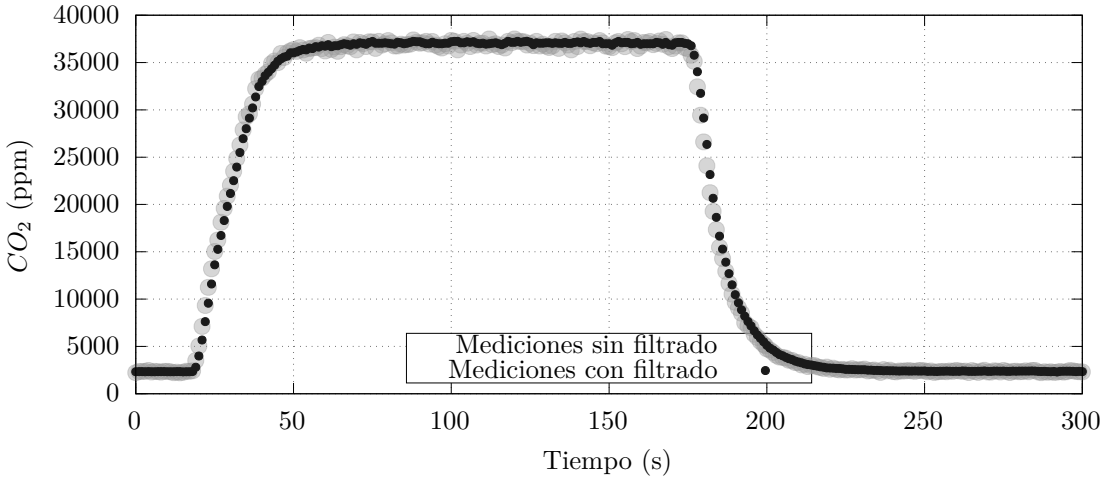


Figura A.20: Gráfica de las mediciones de CO_2 durante la "prueba no. 5" de medición de gases de exhalación.

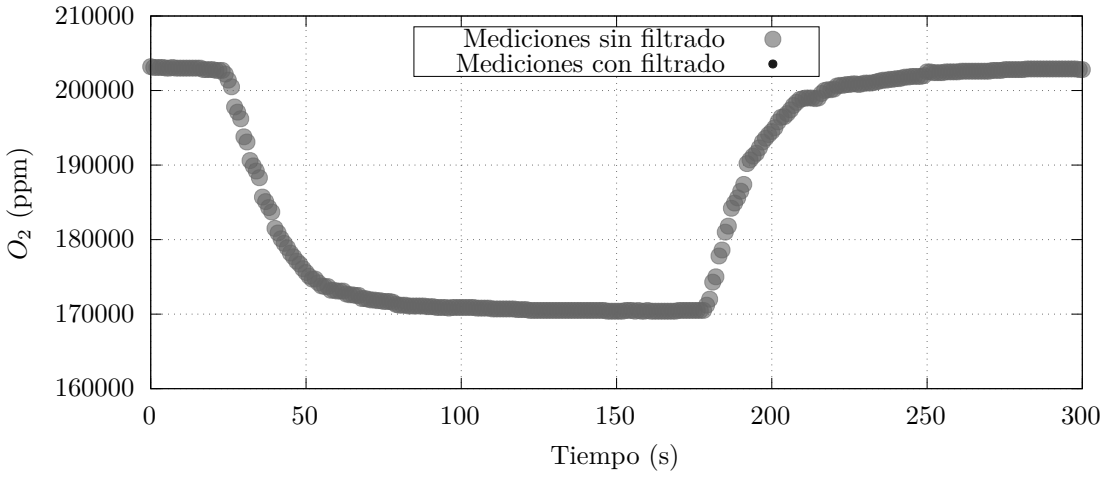


Figura A.21: Gráfica de las mediciones de O_2 durante la "prueba no. 5" de medición de gases de exhalación.

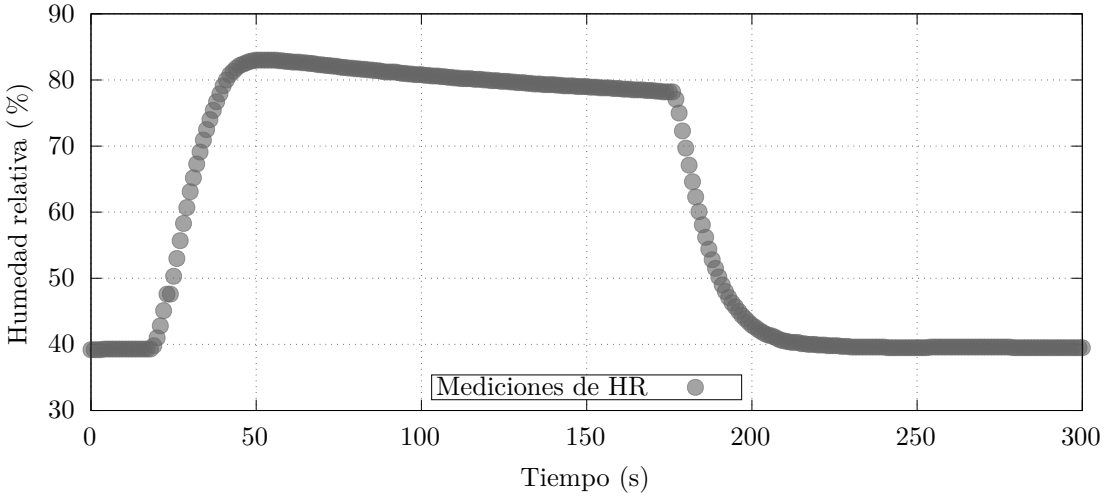


Figura A.22: Gráfica de las mediciones de humedad relativa durante la “prueba no. 5” de medición de gases de exhalación.

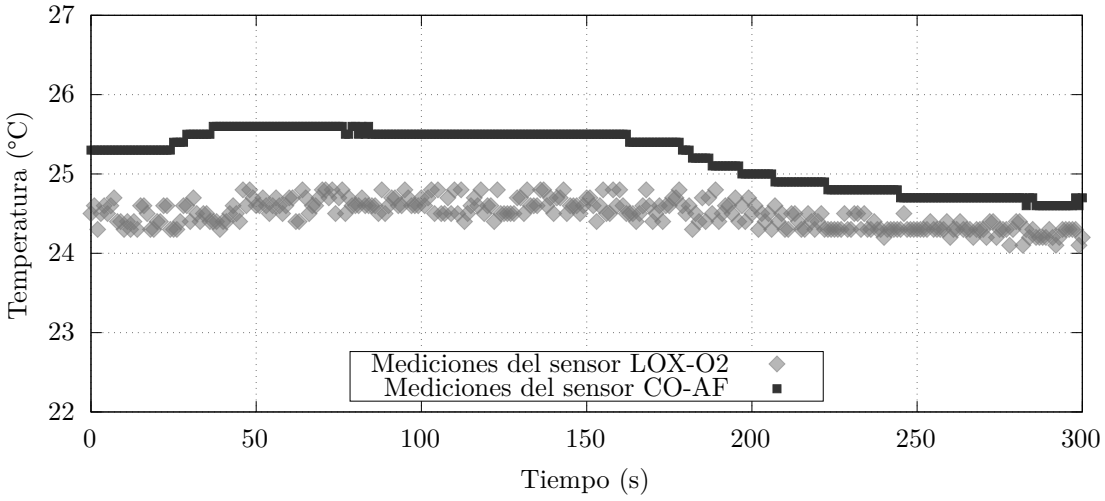


Figura A.23: Gráfica de las mediciones de temperatura durante la “prueba no. 5” de medición de gases de exhalación.

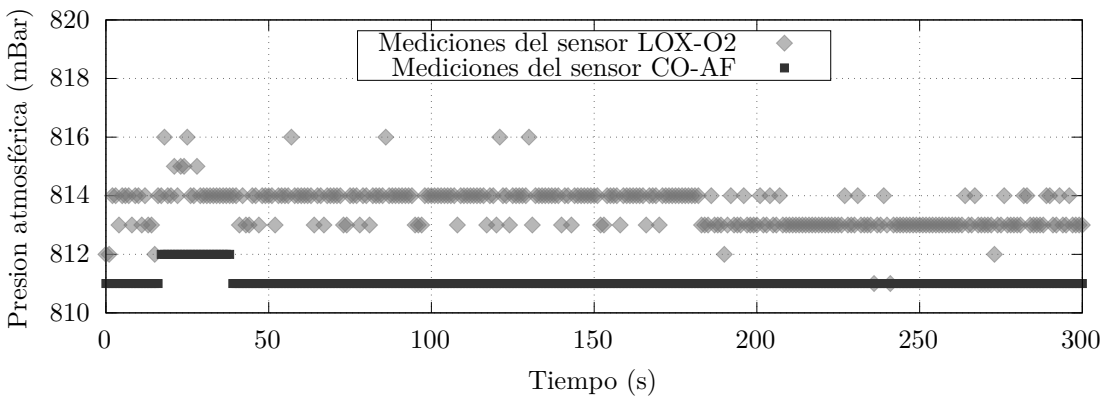


Figura A.24: Gráfica de las mediciones de presión durante la “prueba no. 5” de medición de gases de exhalación.

A.3.5 Adquisición del proceso de combustión: Prueba no. 2

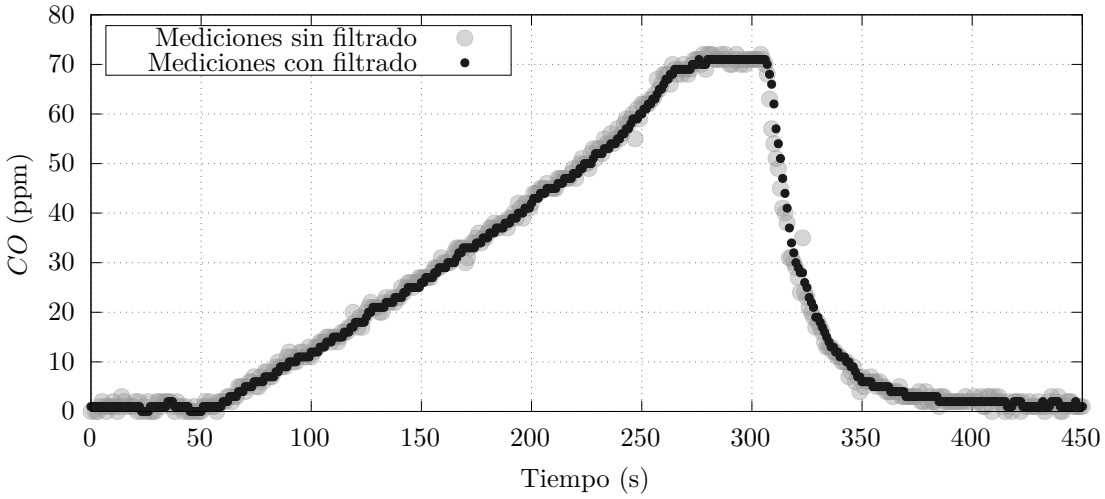


Figura A.25: Gráfica de las mediciones de CO durante la "prueba no. 2" de adquisición del proceso de combustión.

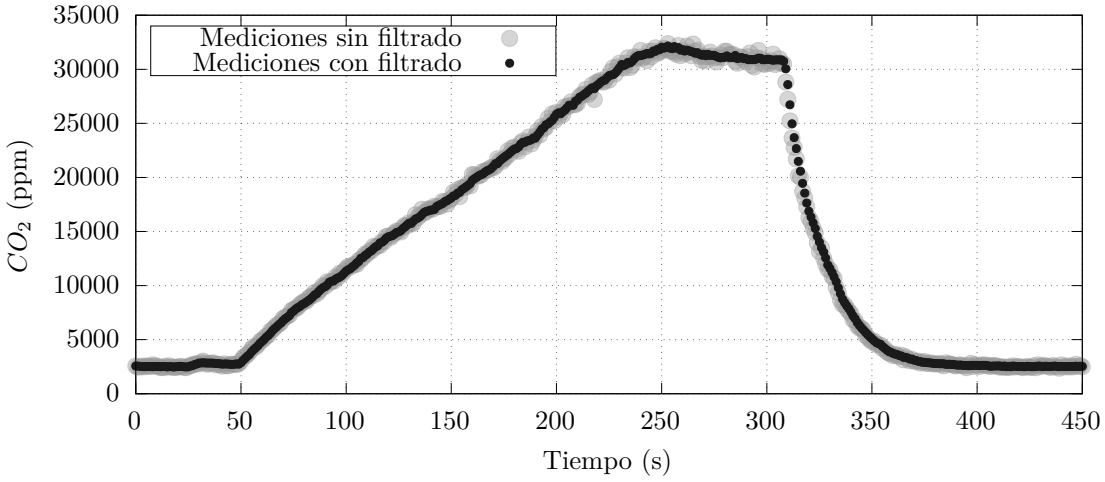


Figura A.26: Gráfica de las mediciones de CO_2 durante la "prueba no. 2" de adquisición del proceso de combustión.

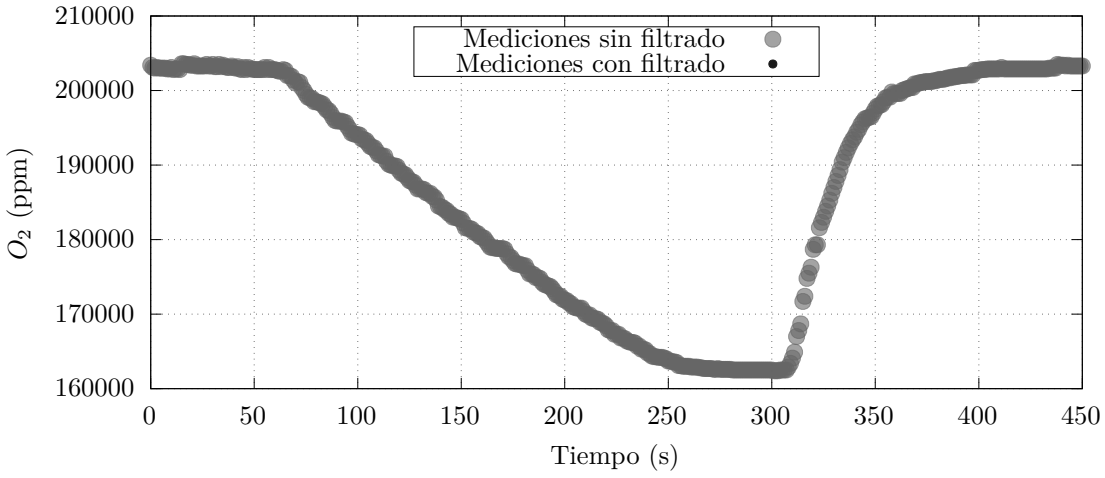


Figura A.27: Gráfica de las mediciones de O_2 durante la "prueba no. 2" de adquisición del proceso de combustión.

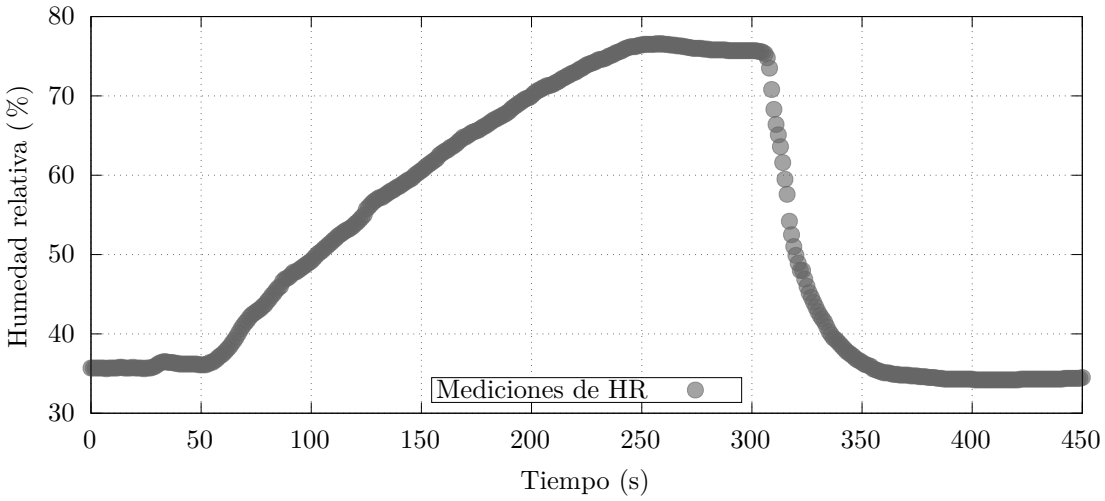


Figura A.28: Gráfica de las mediciones de humedad relativa durante la "prueba no. 2" de adquisición del proceso de combustión.

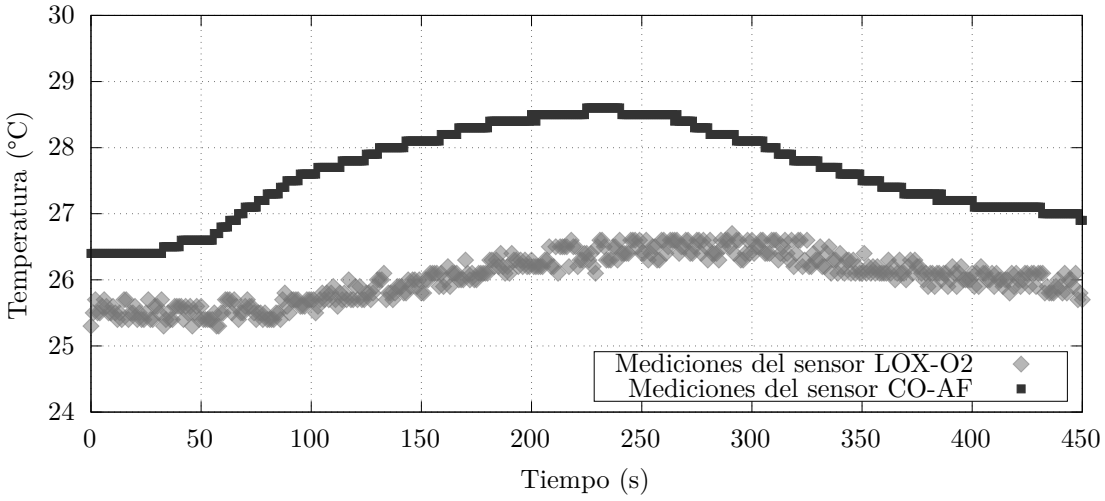


Figura A.29: Gráfica de las mediciones de temperatura durante la "prueba no. 2" de adquisición del proceso de combustión.

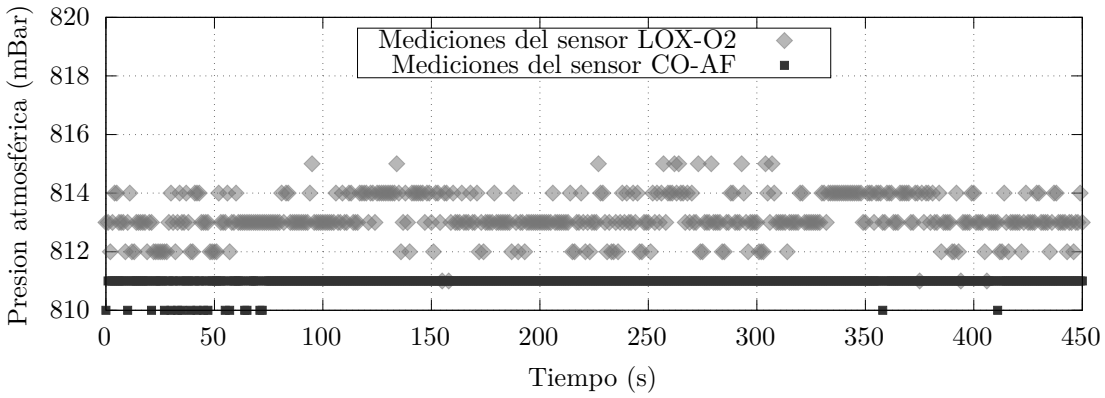


Figura A.30: Gráfica de las mediciones de presión atmosférica durante la “prueba no. 2” de adquisición del proceso de combustión.

A.3.6 Adquisición del proceso de combustión: Prueba no. 3

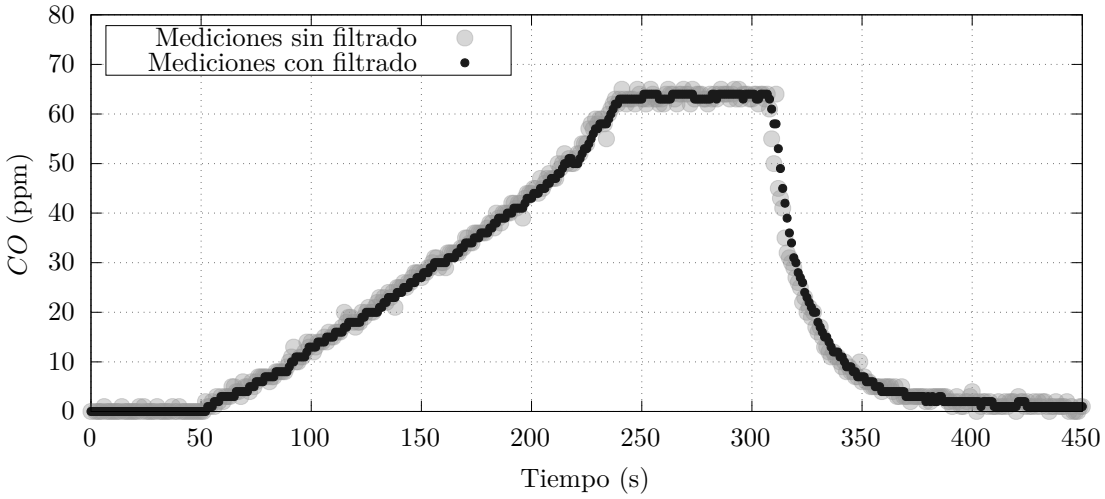


Figura A.31: Gráfica de las mediciones de CO durante la "prueba no. 3" de adquisición del proceso de combustión.

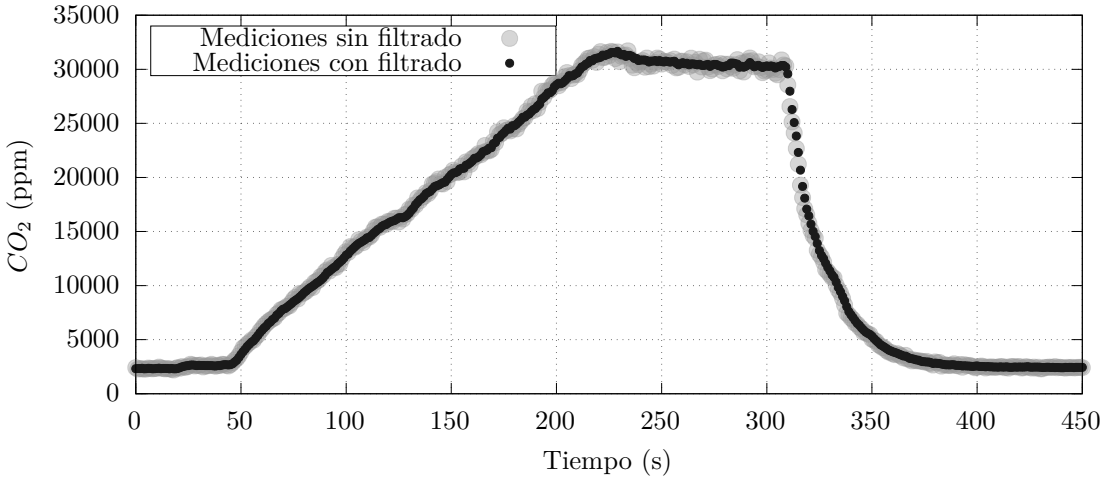


Figura A.32: Gráfica de las mediciones de CO_2 durante la "prueba no. 3" de adquisición del proceso de combustión.

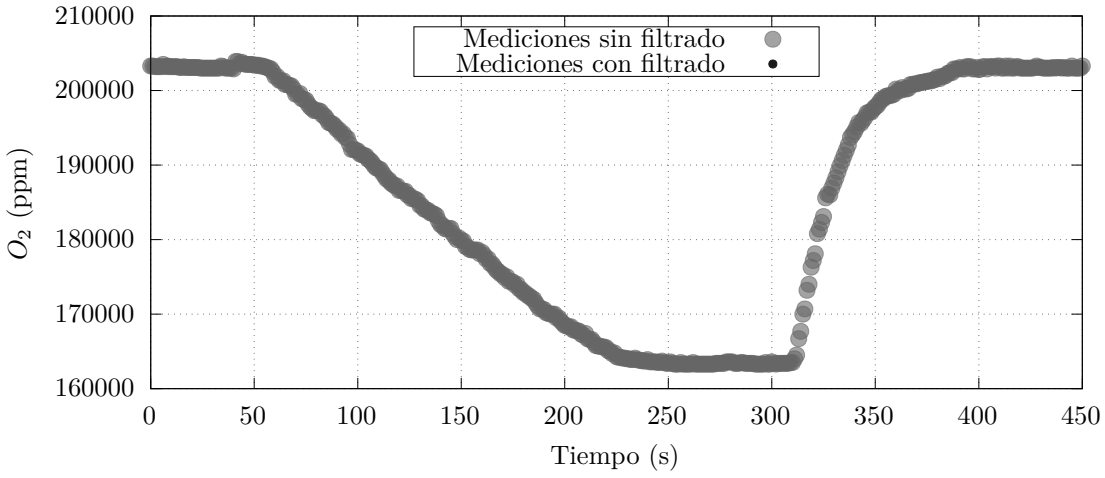


Figura A.33: Gráfica de las mediciones de O_2 durante la "prueba no. 3" de adquisición del proceso de combustión.

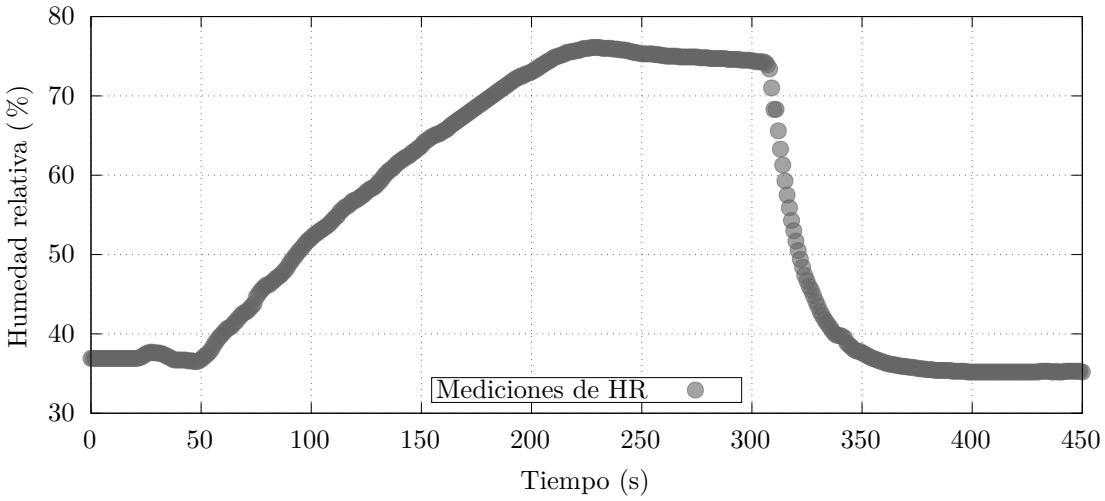


Figura A.34: Gráfica de las mediciones de humedad relativa durante la "prueba no. 3" de adquisición del proceso de combustión.

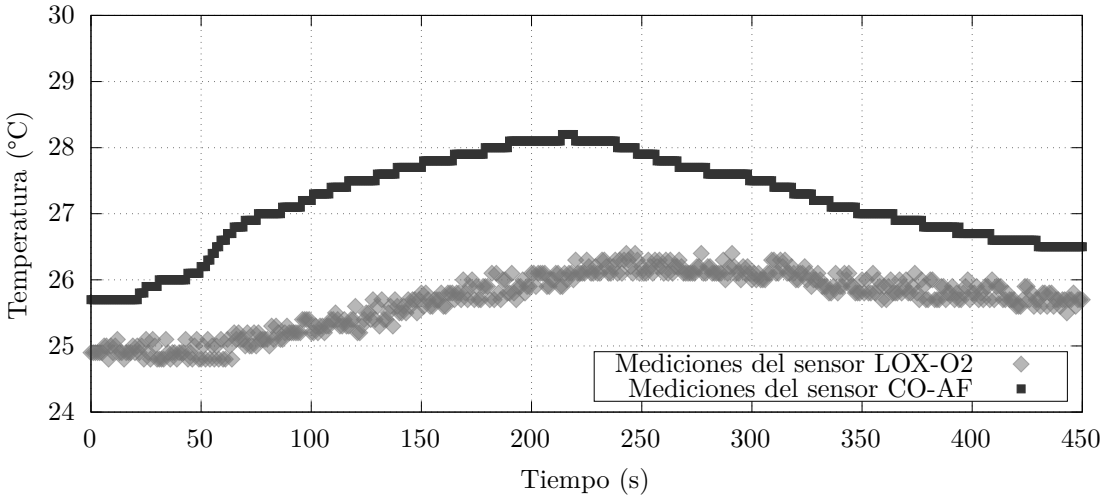


Figura A.35: Gráfica de las mediciones de temperatura durante la "prueba no. 3" de adquisición del proceso de combustión.

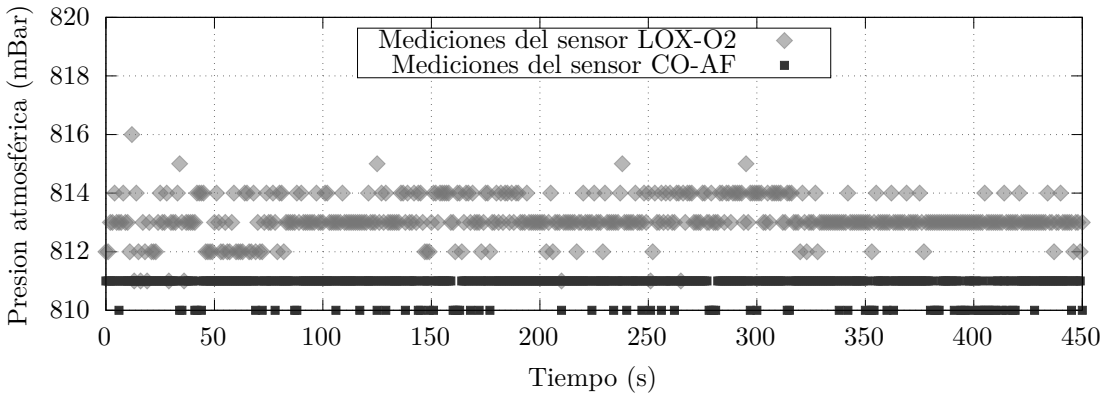


Figura A.36: Gráfica de las mediciones de presión atmosférica durante la “prueba no. 3” de adquisición del proceso de combustión.

A.3.7 Adquisición del proceso de combustión: Prueba no. 4

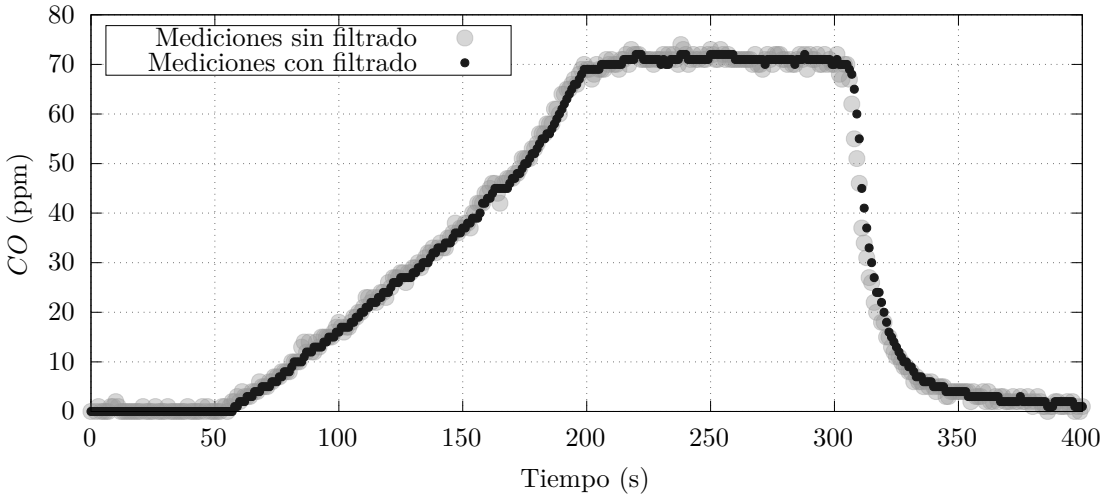


Figura A.37: Gráfica de las mediciones de CO durante la "prueba no. 4" de adquisición del proceso de combustión.

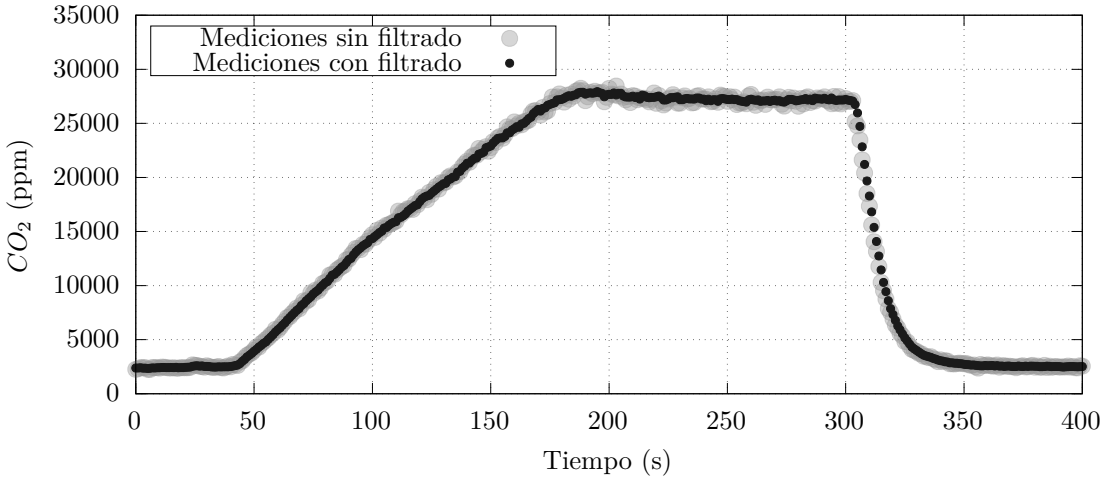


Figura A.38: Gráfica de las mediciones de CO_2 durante la "prueba no. 4" de adquisición del proceso de combustión.

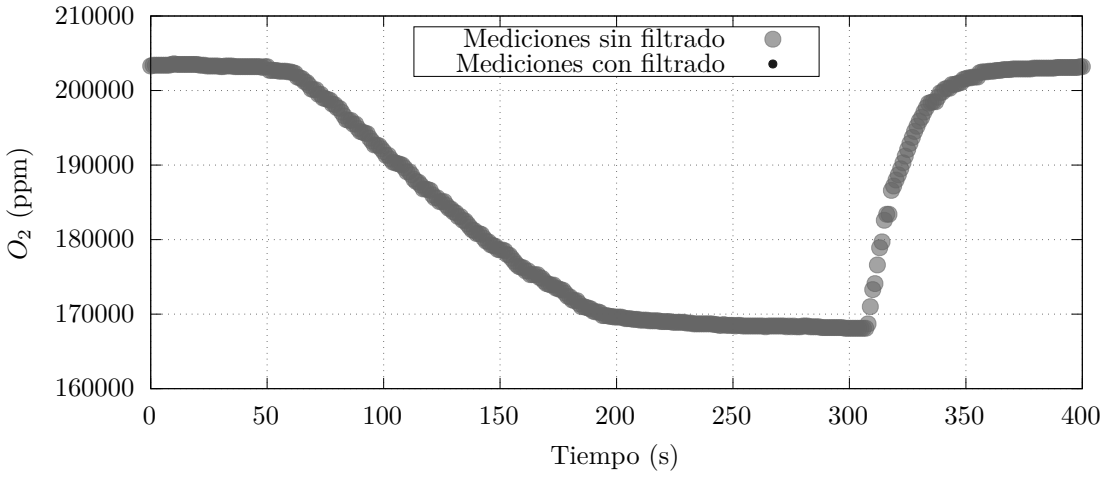


Figura A.39: Gráfica de las mediciones de O_2 durante la "prueba no. 4" de adquisición del proceso de combustión.

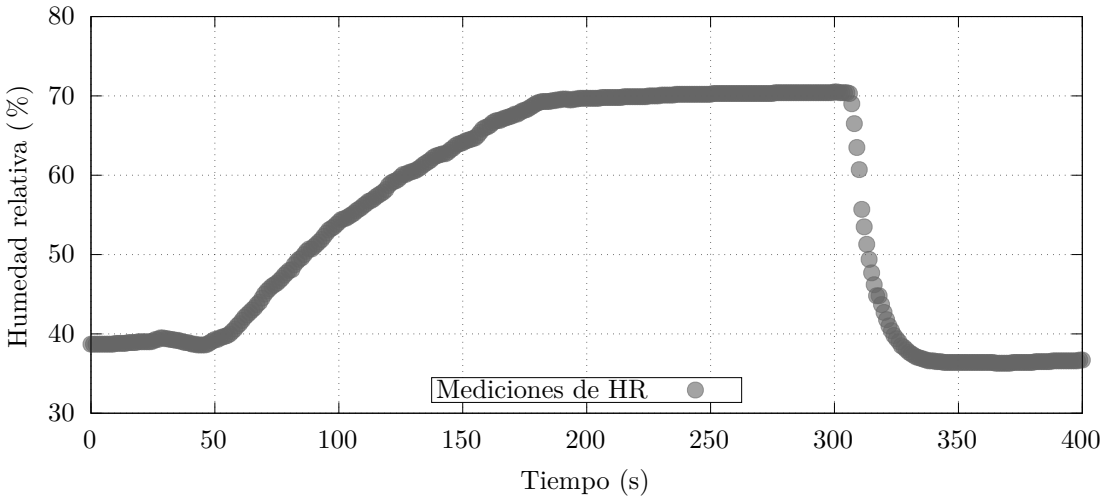


Figura A.40: Gráfica de las mediciones de humedad relativa durante la "prueba no. 4" de adquisición del proceso de combustión.

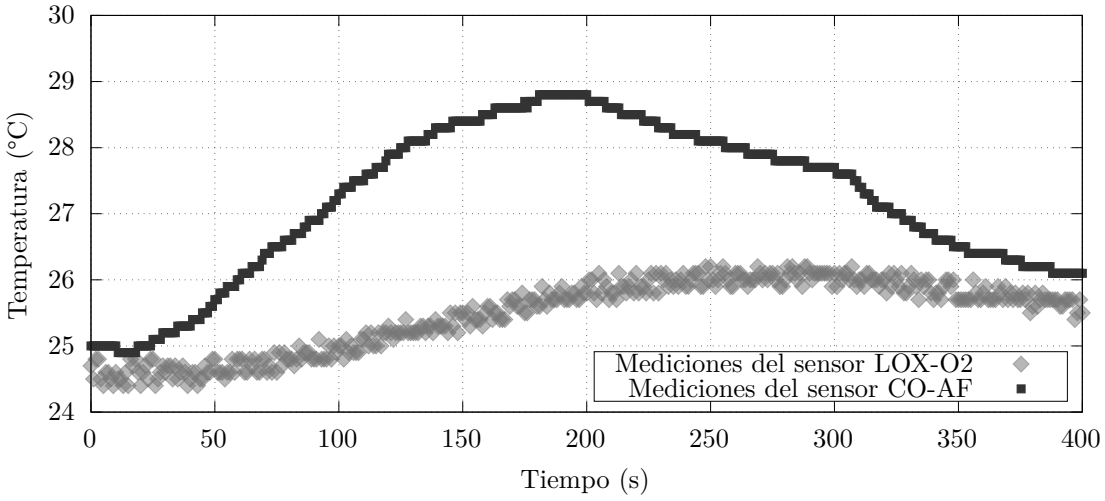


Figura A.41: Gráfica de las mediciones de temperatura durante la "prueba no. 4" de adquisición del proceso de combustión.

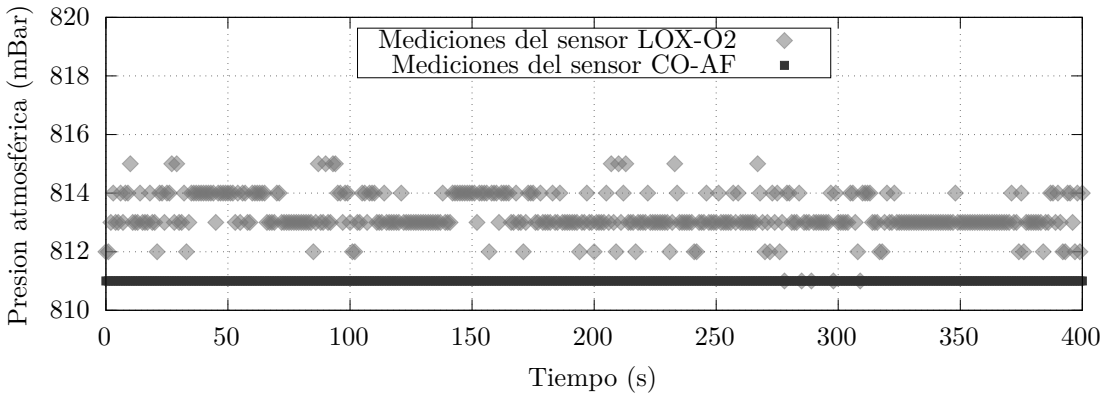


Figura A.42: Gráfica de las mediciones de presión atmosférica durante la “prueba no. 4” de adquisición del proceso de combustión.

A.3.8 Adquisición del proceso de combustión: Prueba no. 5

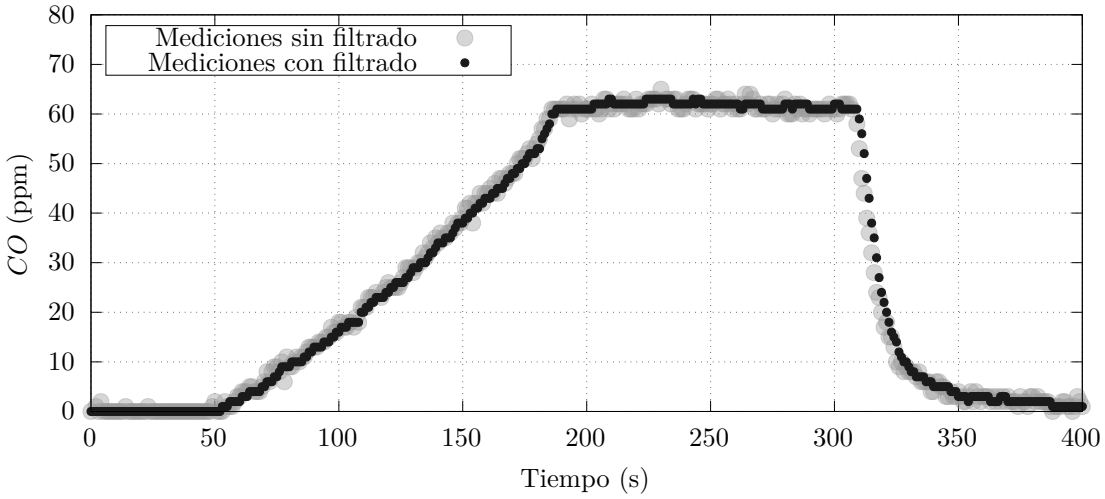


Figura A.43: Gráfica de las mediciones de CO durante la "prueba no. 5" de adquisición del proceso de combustión.

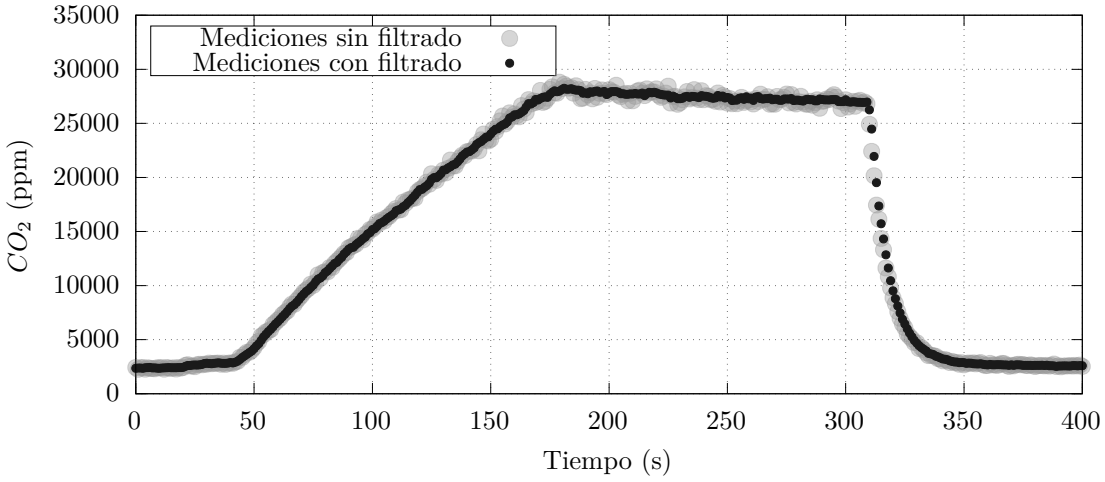


Figura A.44: Gráfica de las mediciones de CO_2 durante la "prueba no. 5" de adquisición del proceso de combustión.

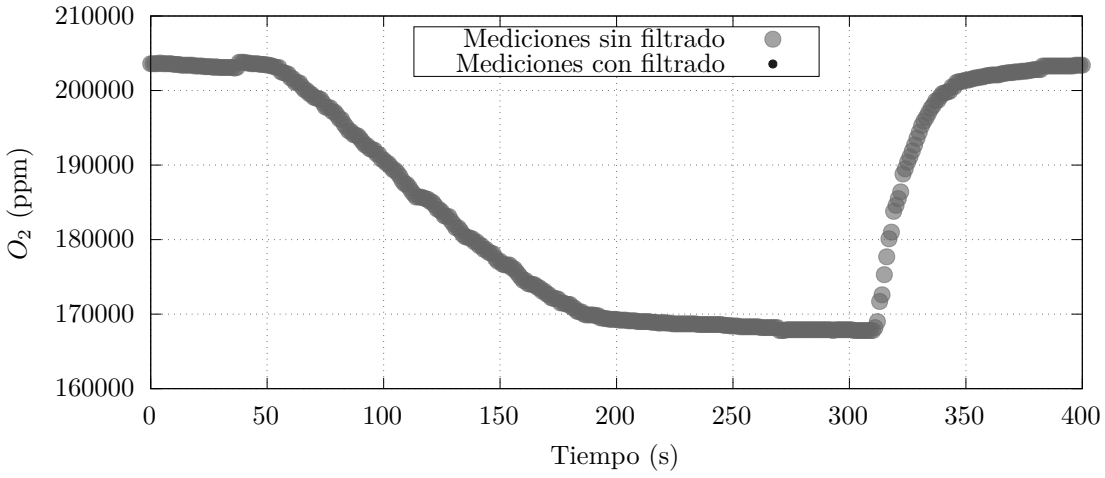


Figura A.45: Gráfica de las mediciones de O_2 durante la "prueba no. 5" de adquisición del proceso de combustión.

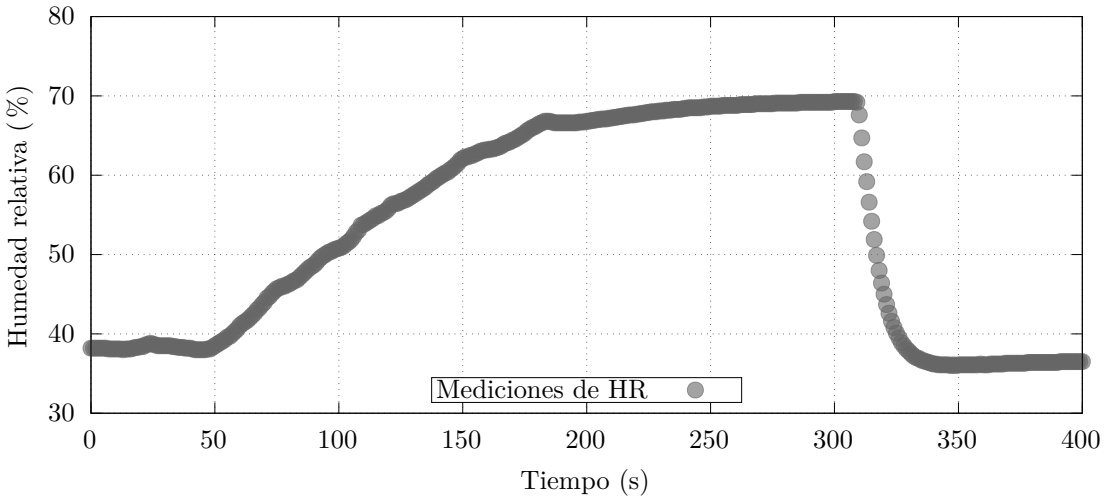


Figura A.46: Gráfica de las mediciones de humedad relativa durante la "prueba no. 5" de adquisición del proceso de combustión.

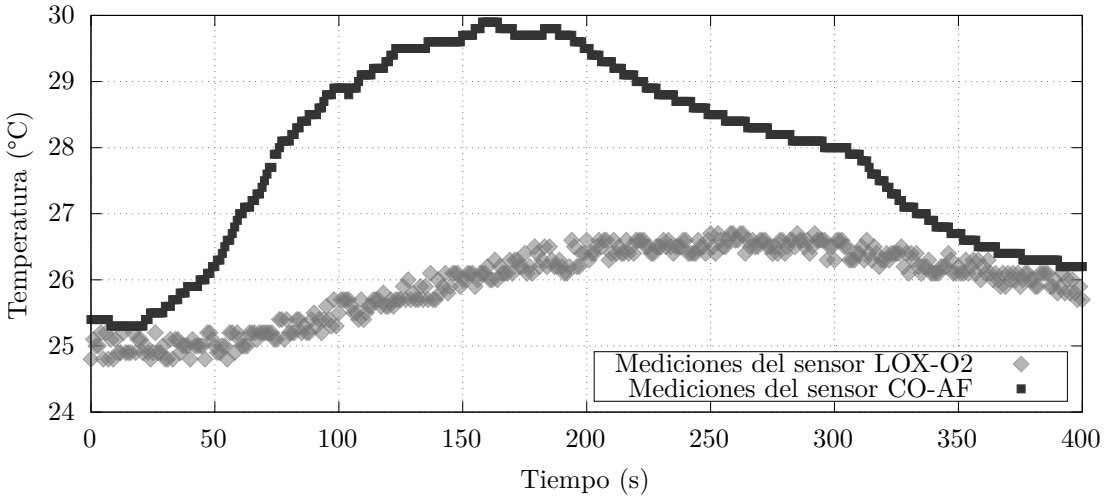


Figura A.47: Gráfica de las mediciones de temperatura durante la "prueba no. 5" de adquisición del proceso de combustión.

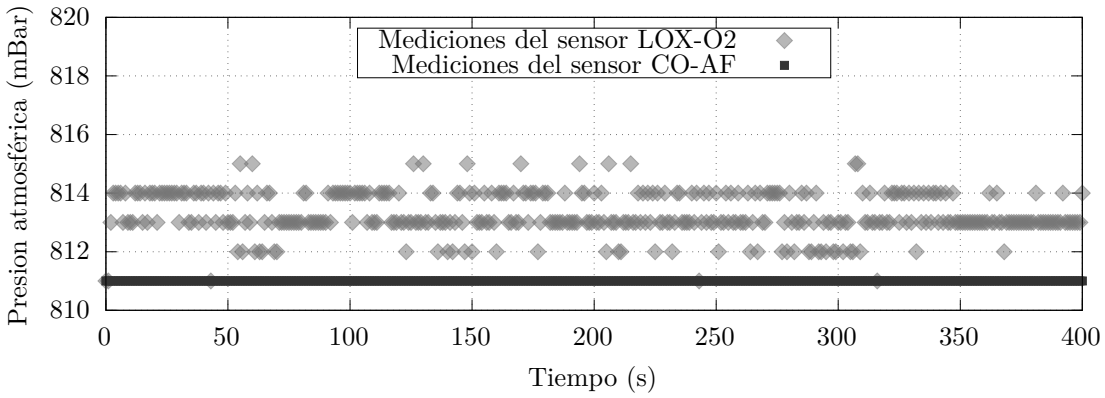


Figura A.48: Gráfica de las mediciones de presión atmosférica durante la “prueba no. 5” de adquisición del proceso de combustión.

Bibliografía

[1] "World steel in figures 2020," 2020.

[2] W. S. A. AISBL. About steel. [Online]. Available: https://www.worldsteel.org/about-steel.html

[3] B. de Economía y Finanzas Bankinter. ¿por qué es tan importante el acero? [Online]. Available: https://www.bankinter.com/blog/economia/acero-aplicaciones-uso

[4] Latem Industries Limited, "Removing mill scale from steel surfaces," jan 2024.

[5] D. J. Young, *High-Temperature Oxidation and Corrosion of Metals*. Oxford, UK: Elsevier, 2016.

[6] SV, Miss Pulate and AB, Mrs Diggikar, "Embedded web server based interactive data acquisition and control system."

[7] T. Araari, L. Charaabi, and K. Jelassi, "Design and implementation of an embedded data acquisition based on linux system for smart grid," in *2014 International Conference on Electrical Sciences and Technologies in Maghreb (CISTEM)*. IEEE, 2014, pp. 1–5.

[8] A. M. Anand, B. Raveendran, S. Cherukat, and S. Shahab, "Using pruss for real-time applications on beaglebone black," in *Proceedings of the Third International Symposium on Women in Computing and Informatics*. ACM, 2015, pp. 377–382.

[9] B. Travaglione, A. Munyard, and D. Matthews, "High resolution undersea acoustic data acquisition using single-board microcontrollers," 2015.

[10] F. A. Rodríguez Corbo, A. Hernández González, and J. Ramírez Beltrán, "Adquisición de datos analógicos con alta precisión usando una computadora de placa única," *Ingeniería Electrónica, Automática y Comunicaciones*, vol. 39, no. 3, pp. 68–76, 2018.

[11] Dr. R. Lakshmi Narayana, Prof. K. Nagabhushan Raju, "Development of open source real time data acquisition system," *International Journal of Applied Engineering Research*, vol. 14, no. 8, pp. 2038–2042, 2019.

[12] N. Khera, P. Sharma, D. Shukla, and I. G. Dar, "Development of web based gas monitoring system using labview," in *2017 Recent Developments in Control, Automation & Power Engineering (RDCAPE)*. IEEE, 2017, pp. 439–441.

[13] National Instruments Corporation. (2020) ¿qué es adquisición de datos? [Online]. Available: http://www.ni.com/data-acquisition/what-is/esa/

[14] Revista Electro Industria. (2018) ¿qué son los sistemas daq? [Online]. Available: http://www.emb.cl/electroindustria/articulo.mvc?xid=3286&edi=164&xit=que-son-los-sistemas-daq

[15] D. Molloy, *Exploring BeagleBone: tools and techniques for building with embedded Linux.* John Wiley & Sons, 2019.

[16] C. Hallinan, *Embedded Linux primer: a practical real-world approach.* Pearson Education India, 2011.

[17] Software in the Public Interest. (2020) Acerca de debian. [Online]. Available: https://www.debian.org/intro/about

[18] Free Software Foundation, Inc. (2020) Acerca de debian. [Online]. Available: https://www.gnu.org/

[19] T. Ylonen. (2020) Ssh (secure shell). [Online]. Available: https://www.ssh.com/ssh/

[20] L. M. C. P. Rui Santos. How to launch the cloud9 ide on your beaglebone. [Online]. Available: https://www.dummies.com/computers/beaglebone/how-to-launch-the-cloud9-ide-on-your-beaglebone/

[21] M. Kerrisk and P. Zijlstra, "Linux programmer's manual," *Linux Man-Pages Project*, 2017.

[22] L. Jones, "Wg14 n1539 committee draft iso/iec 9899: 201x," 2010.

[23] P. Worsfold, A. Townshend, C. F. Poole, and M. Miró, *Encyclopedia of analytical science.* Elsevier, 2019.

[24] SST SENSING LTD, "Luminox o2 sensors data sheet"," 2017.

[25] Alphasense Ltd, "Co-af carbon monoxide sensor technical specification," 2017.

[26] CO2, Meter, "Ec200 electrochemical sensor controller manual," 2019.

[27] Gas Sensing Solutions, "Sprintir," 2020.

[28] CO2 Meter, "Sprintir datasheet," 2016.

[29] Texas Instruments, "Isow784x high-performance, 5000-vrms reinforced quad-channel digital isolators with integrated high-efficiency, low-emissions dc-dc converter datasheet (rev. f)," 2019.

[30] A. K. Anand Reghunathan, Koteshwar Rao, "Low-emission designs with isow7841 integrated signal and power isolator," 2019.

Colophon

This document was typeset using the `itmthesis` class developed by Gerardo Marx Chávez-Campos. The class was designed based on the `classicthesis` class developed by André Miede. The `itmthesis` is available for LaTeX in a Bitbucket repository:

https://gmarxcc@bitbucket.org/itmthesis/the-itmorelia-thesis-class.git,

or in a Git-Hub repository:

https://github.com/gmarxcc/the-itmorelia-thesis-class

Happy users of `itmthesis` class may send a real postcard, coffee mug, hoody or any kind of present to my university "Instituto Tecnológico de Morelia", Tecnologico Avenue, Morelia city, Michoacan State, Mexico.

Colofón

Este documento fue generado usando el estilo desarrollado en la clase `itmthesis`, desarrollada por Gerardo Marx Chávez-Campos. Esta fue diseñada basada en el estilo `classicthesis` desarrollado por André Miede.

El estilo `itmthesis` y un ejemplo de su uso está disponible en repositorios para ser compilados en LaTeX en Bitbucket:

https://gmarxcc@bitbucket.org/itmthesis/the-itmorelia-thesis-class.git,

o en Git-Hub:

https://github.com/gmarxcc/the-itmorelia-thesis-class

Si eres un usuario agradecido con este diseño puedes mandar una postal, taza de café, sudadera de tu universidad o cualquier tipo de presente en forma de agradecimiento a la universidad de origen de tu servidor: "Instituto Tecnológico de Morelia", Av. Tecnológico #1500, Col. Santiaguito, Morelia, Michoacán, México.

Declaración

Yo, **Edwars Sayeth Rodríguez Martínez**, con número de control **13121635** declaro que el trabajo titulado: *Desarrollo de un sistema de adqusición de datos basado en Linux embebido para la medición de concentraciones de gases en reacciones sólido gas,* , es resultado de mi trabajo e investigación original. No se ha realizado una copia de otro trabajo o fuente excepto los respectivamente citados de forma explícita en el texto.

Morelia, Michoacán, México, Mes 2025

Edwars Sayeth Rodríguez
Martínez, 5 de marzo de 2025