UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática y Matemáticas

TRABAJO FIN DE GRADO

Sistema miniaturizado de mediciones de dióxido de carbono

Autor: Patricia González Aparicio
Tutor: Stefan Palzer

junio 2019

Patricia González Aparicio
Sistema miniaturizado de mediciones de dióxido de carbono

Patricia González Aparicio

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Agradecimientos

En primer lugar me gustaría agradecer a mi tutor Stefan Palzer por su ayuda y apoyo en todo momento para crear este TFG. A mi compañero de despacho recién doctorado Álvaro Ortiz por explicarme todas las dudas que pudiera tener y a Daniel Serena por aconsejarme y acompañarme en este duro año. No podría haber tenido más suerte con mis compañeros y "jefe".

En particular quiero destacar el apoyo y confianza que he recibido por parte de mi familia desde pequeña, especialmente mis padres Patricia Aparicio y Alberto González. Si no fuera por ellos no habría cursado esta carrera ni habría llegado hasta aquí. Siempre pensando que podía con todo y más.

También quiero tener un recuerdo para mis compañeros de clase, todos ellos muy diferentes pero igual de especiales. El ser un grupo tan pequeño nos ha permitido ser cercanos los unos con los otros y hemos podido conocer hasta el último rasgo de personalidad de cada uno en estos cinco años de carrera.

Por último, pero nunca menos importante, mis amigos. Porque no los hay mejores y los quiero un montón.

Resumen

Los llamados gases traza juegan un papel importante en el comportamiento del clima regional y global. Uno de los más importantes es el dióxido de carbono por su relación con el cambio climático. Sin embargo, en la actualidad, no existen sistemas capaces de tomar datos fiables sobre su flujo y concentración a escalas relevantes tanto espaciales como temporales. El objetivo principal de mi TFG es el diseño, desarrollo y caracterización de un sensor de dióxido de carbono lo suficientemente pequeño y ligero como para poder ser integrado en un Dron utilizando una Raspberry Pi. De esta manera se podrá medir la concentración de dicho gas en diversas posiciones para posteriormente poder modelar su movimiento y detectar la evolución de su flujo y concentración. .

Con el fin de obtener un conjunto de mediciones más completo, se han integrado también sensores de temperatura, humedad y presión barométrica en el sistema. Estos sensores son de carácter comercial y utilizan el protocolo de comunicación Inter-Integrated Circuit, por lo que se ha implementado un módulo para cada uno de ellos con las funciones y comandos necesarios para su configuración y gestión. El sensor de dióxido de carbono es de diseño y desarrollo propio y se basa en la detección fotoacústica. Está compuesto por diferentes dispositivos que, mediante la conversión de una onda de formato digital a analógico, permiten que se emita luz y ésta sea recogida por un micrófono y convertida de nuevo a digital. Para ello se han utilizado los protocolos de comunicación Serial Peripheral Interface e Inter-Integrated Circuit y se ha implementado una librería que permite la comunicación con cada uno de los componentes de este sensor. Por otro lado, con un conversor analógico digital se modela la señal recibida por el micrófono, la cual está relacionada directamente con la concentración de dióxido de carbono y, para el estudio de la amplitud de la onda, se han implementado los algoritmos Fast Fourier Transform y Goertzel. Se ha hecho, por tanto, un estudio de los límites y restricciones de cada componente. Todo ello ha sido desarrollado bajo una interfaz cliente-servidor en la que el cliente solicita el comienzo de la toma de medidas y el servidor ejecuta el mandato. Todos los sensores han sido testeados y validados conformando un sistema de mediciones completo, diseñado y desarrollado con el objetivo de que se incorpore en un dron.

Palabras clave

Dióxido de carbono, sensor miniaturizado, dron, concentración, medida, detección fotoacústica, protocolo de comunicación, emisor, receptor, absorción, amplitud, frecuencia.

Abstract

The so-called trace gases play an important role in the behavior of the regional and global climate. One of the most important is carbon dioxide because of its relationship with climate change. However, nowadays, there are no systems capable of taking reliable data on their flow and concentration at relevant scales, both spatial and temporal. The main objective of my TFG is the design, development and characterization of a carbon dioxide sensor small and light enough to be integrated into a Drone using a Raspberry Pi. In this way, it will be possible to measure the concentration of this gas in several locations in order to be able to model its movement and detect the evolution of its flow and concentration. Through the use of the photoacoustic principle, the sensor achieved reaches a higher resolution than that provided by the current sensors.

In order to obtain a more complete set of measurements, temperature, humidity and barometric pressure sensors have also been integrated into the system. These sensors are commercial and use the Inter-Integrated Circuit communication protocol, so a module has been implemented for each of them with the functions and commands necessary for configuration and management. The carbon dioxide sensor has been designed and developed by my own. It is made up of different devices that, through the conversion of a wave from digital to analog format, allow light to be emitted and it is picked up by a microphone and converted back to digital. To this end, the communication protocols Serial Peripheral Interface and Inter-Integrated Circuit have been used, so that, like the other sensors, a library has been implemented that allows communication with each of the components of this sensor. A study has been made of the limit frequency capable of reaching the analogue digital converter used, and the relationship between this and the diode that emits light. On the other hand, with a digital analog converter the signal received by the microphone is modeled, which is directly related to the concentration of carbon dioxide and, for the study of the amplitude of the wave, the Fast Fourier Transform and Goertzel algorithms have been implemented. All this has been developed under a client-server interface in which the client requests the beginning of the taking of measurements and the server executes the command. All the sensors have been tested and validated, forming a complete measurement system, designed and developed with the aim of incorporating it into a drone.

Keywords

Carbon dioxide, miniaturized sensor, drone, concentration, measurment, photoacoustic detection, communication protocol, trasmitter, receiver, absorption, amplitude, frecuency

ÍNDICE

Listas

Lista de códigos

Lista de ecuaciones

Lista de figuras

Lista de tablas

1

Introducción

La atmósfera está compuesta por diferentes gases como el oxígeno, nitrógeno, argón, vapor de agua y dióxido de carbono entre otros. Especialmente la concentración de dióxido de carbono (CO_2) en la atmósfera esá en constante variación. Un 16 % de la superifice terrestre está cubierta por permafrost (vegetación congelada durante la última edad de hielo, 20.000 años) que contiene una cantidad de carbono similar a la existente en toda la atmósfera actualmente [1]. Debido al calentamiento global, las capas de hielo que cubren el permafrost se están derritiendo, convirtiendo todo el carbono guardado en gases de efecto invernadero como el CO_2. Es por ello por lo que es necesario diseñar un sistema de medidas de CO_2 que ayude estudiar y modelar la fluctuación de este gas a larga escala temporal y espacial. Este gas tiene influencia tanto en el efecto inverandero como en la alimentación de las plantas de manera significativa. Una variación importante en la concentración de este gas puede provocar cambios en la estructura, desarrollo y función de estas y, por lo tanto, afectar a los cultivos. Por lo que tener un control sobre el CO_2 en las cosechas es fundamental [2].

Para poder medir la fluctuación en la concentración de CO_2 se ha desarrollado un sensor miniaturizado de bajo coste, tamaño y peso con el fin de poder integrarlo en un dron y tomar medidas en las localizaciones pertinentes de forma sencilla y dinámica. Hasta ahora podemos encontrar diferentes dispositivos capaces de medir simultáneamente CO_2, temperatura y humedad. Estos usan un detector de gas óptico infrarrojo de baja resolución y microsensores de temperatura y humedad [3]. Sin embargo, estos sistemas tienen un tamaño bastante grande, son caros, y su función es limitada. El objetivo de este proyecto es diseñar y desarrollar un sistema que monitoriza la toma de medidas de CO_2, temperatura, humedad y presión, todo ello integrado en una miniplaca. El sistema está formado por elementos de bajo coste, siendo su núcleo un microsensor de CO_2 de diseño e implementación propia basado en detección fotoacústica. Todo ello se ha realizado bajo la interfaz cliente-servidor para que las medidas sean enviadas a una aplicación base que las estudie y modele.

Este TFG está estructurado en 5 capítulos, empieza con el capítulo 2 con el Estado del Arte. En este capítulo se presenta de forma detallada los dispositivos actuales de medición de CO_2, temperatura, humedad y presión.

El diseño del sistema se muestra en el capítulo 3. En él se explica el funcionamiento y gestión de

los sensores de temperatura, humedad y presión, todos ellos manejados mediante el protocolo de comunicación Inter-Integrated Circuit (I^2C) ; así como el diseño del sensor de CO_2 y el papel que juegan todos los componentes que lo conforman: Digital-to-Analog Conversion (DAC) , Voltage Controlled Current Source (VCCS) , Light-emitting diode (LED) , micrófono Microelectromechanical System (MEMS) , amplificador y Analog-to-Digital Conversion (ADC) . Adicionalmente, se explican los algoritmos Fast Fourier Transform (FFT) y Goertzel utilizados para el posterior análisis de datos, necesario para la obtención de los datos de concentración de CO_2. Y, finalmente, se expone de manera breve la interfaz cliente-servidor del sistema.

En el capítulo 4 se hace un estudio de los componentes más importantes que conforman el sensor de CO_2, sus limitaciones y los parámetros de configuración que permiten que los resultados obtenidos sean óptimos. Además, se hace un experimento en el que se exhala aire a una distancia cercana al sensor, midiendo así la variación de concentración de CO_2. Finalmente, en el capítulo 5, se muestran las principales conclusiones y resultados obtenidos, así como posibles cambios que se puedan llevar a cabo en un futuro para mejorar el funcionamiento del sistema.

2

Estado del arte

El incremento de determinados gases de la atmósfera y el efecto que ellos producen ha provocado un aumento en la investigación y desarrollo de dispositivos capaces de medir parámetros físicos y químicos. Dos magnitudes que afectan directamente al comportamiento de estos gases son la temperatura y la humedad. En el caso de los sensores de temperatura, los más usados actualmente son los Resistencia Temperatura Detector (RTD) , termistores, termopares o sensores infrarrojo [4]. En particular, el funcionamiento de los RTDs se basa en la relación casi lineal temperatura-resistencia que presentan algunos materiales. El más común es el platino, utilizado en la mayoría de microsensores de temperatura. Para el cálculo de humedad existen varios tipos: de desplazamiento, psicométrico, polímero resistivo o capacitivo. Concretamente, los capacitivos miden la humedad mediante la variación en la capacidad interna. Tanto los RTDs para la temperatura, como los sensores capacitivos para la humedad, poseen los requisitos de coste, peso, tamaño y resolución para ser integrados en un dron, por lo que serán los que se usen en este proyecto [5]. Por otro lado, los sensores de presión utilizan una membrana sobre la que hay cuatro resistencias que varían según la presión mecánica. Típica estructura MEMS de sensores de temperatura, humedad y presión se muestra en la figura 2.1.

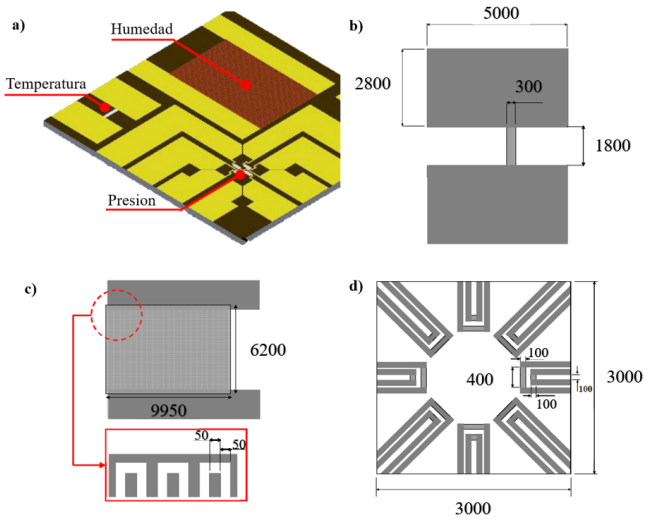


Figura 2.1: Configuración y dimensiones de sensores RTD (b), humedad (c) y presión (d). Unidades μm. [6]

Uno de los gases que tiene especial importancia es el CO_2. Por ello, se han creado diferentes sensores que traducen información química en impulsos eléctricos en respuesta a cambios en la composición del gas. De acuerdo con International Union of Pure and Applied Chemistry (IUPAC) existen siete grupos diferentes de sensores químicos capaces de realizar dicha función: ópticos, electroquímicos, eléctricos, magnéticos, masa-sensibles, termométricos y basados en la radiación [7]. Sin embargo, la mayoría de ellos están limitados por su baja eficacia y resolución, o su coste y tamaño. Por ejemplo, los sensores electroquímicos tardan mucho en medir, los magnéticos no son eficientes y los masa sensibles son muy caros. Lo más adecuado es usar sensores ópticos basados en la absorción infrarroja. Los más destacados se encuentran en la tabla 2.1.

El uso del espectro de absorción del CO_2 es una opción de bajo coste que permite determinar la concentración de este gas. Los sensores Non-dispersive infrared spectroscopy (NDIR) se basan en ello, aportando una alternativa sencilla y de alta calidad. El principio de atenuación de la luz según la ley de Beer-Lambert proporciona una medida de absorción de luz de las partículas de CO_2 [8]:

$$A(\lambda, n, Z) = I_0 e^{-\sigma(\lambda)NZ} \tag{2.1}$$

donde λ es la longitud de onda del CO_2, N la concentración de dicho gas, Z el camino óptico, I_0 es la intensidad de la luz medida en una cámara vacía y $\sigma(\lambda)$ el área de absorción de una molécula de CO_2. Este principio facilita una manera de diseñar sistemas capaces de identificar diferentes gases y su concentración. Adicionalmente, cumple con los requisitos de resolución, peso y coste. Es por ello por lo que se ha elegido este principio para diseñar y desarrollar el sensor de CO_2 de este proyecto [9].

Sensor	**GMM222C**	**K30**	**S100**	**AN100**
fabricante	Vaisala	Sense Air	ELT	KCD
LxWxD (mm)	cilindro con 18 mm de diámetro y 140 mm de altura	51 x 57 x 14	33 x 33 x 13	82 x 45 x 18
Peso (g)	220	17	10	29
Rango de medida (ppm)	0-2,000	0-5,000	0-5,000	0-5,000
Precisión	33 ppm + 2 %	30 ppm + 5 %	3 ppm + 5 %	200 ppm + 3 %
Tiempo de respuesta (s)	30 (63 %)	20 (63 %)	60 (90 %)	30 (63 %)
Voltaje de trabajo (V)	11-20 VDC	4.5-14 VDC	5.0-5.5 VDC	8-14 VDC

Tabla 2.1: Tabla con las especificaciones de cuatro sensores de CO_2 NDIR comerciales [10].

Un esquema del funcionamiento de los sensores de CO_2 actuales y el espectro de absorción de dicho gas se muestra en la figura 2.2.

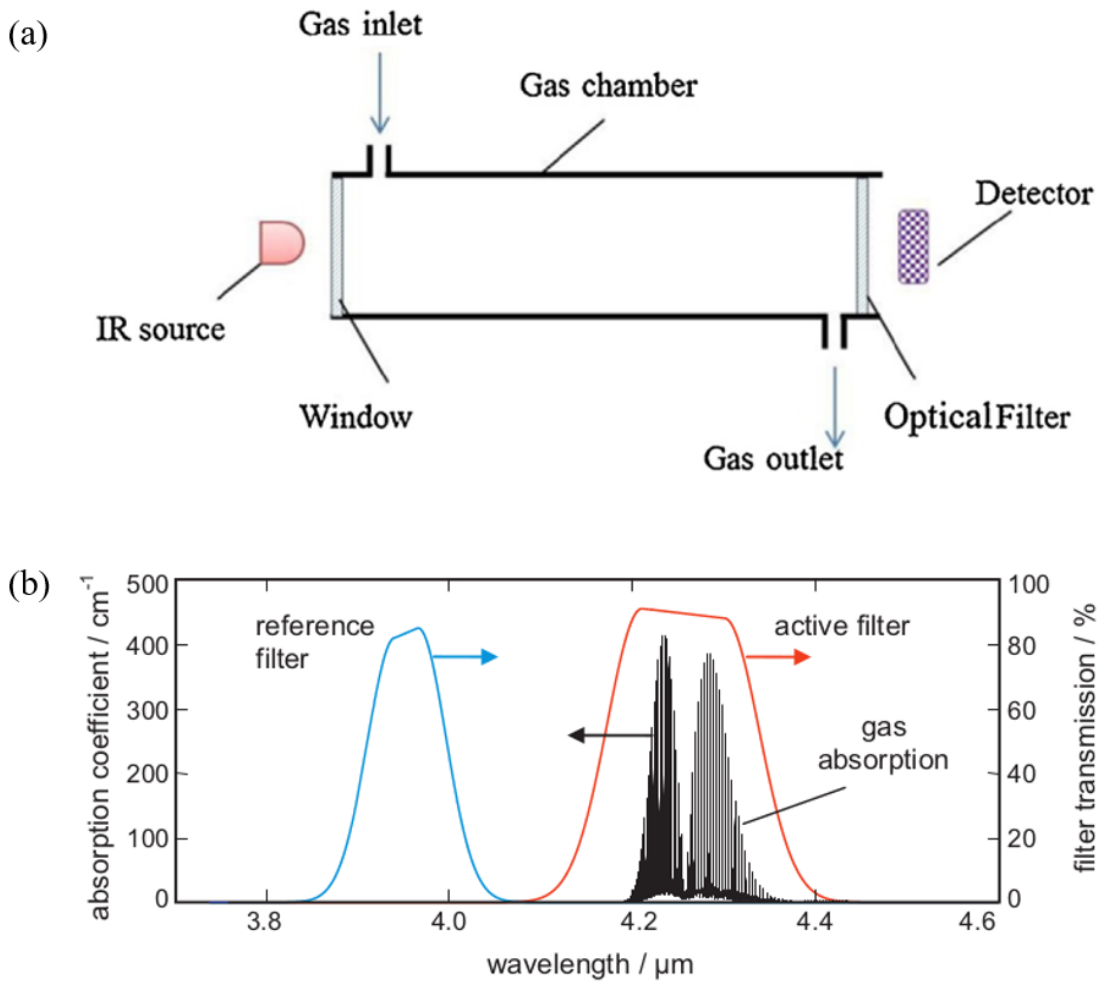


Figura 2.2: La figura (a) muestra el funcionamiento de un sensor NDIR comercial. Esta basado en la absorción de luz infrarroja que presentan las moléculas de algunos gases como el CO_2, CO, NO_2 entre otros. Se emite luz dentro de una cámara por la que pasa gas y se mide la energía que recibe el detector [11]. La figura (b) muestra el espectro de absorción de CO_2 el cual se presenta en longitudes de onda cercanas entre 4.2 y 4.4 μm [12].

3

Diseño del sistema

En este proyecto se ha diseñado un sistema de medición ambiental que incluye sensores comerciales de temperatura y humedad (SHT21 [1], Sensirion), y presión (MPL3115 [2], Semiconductor, Inc.); y un sensor de CO_2 de diseño propio. Todos ellos han sido gestionados por una Raspberry Pi 3 B + [3] mediante los protocolos de comunicación I^2C [13] y Serial Peripheral Interface (SPI) [14]. Así mismo, se ha decidido implementar dicho sistema de mediciones bajo una interfaz cliente-servidor con protocolo de comunicación Transmision Control Protocol (TCP) / Internet Protocol (IP) en la que la aplicación cliente se encontrará en un ordenador base guardando los datos en fichero, mientras que el servidor se ejecutará en la Raspberry, la cual se incluirá en un dron. Una vez que cliente y servidor están conectados, el servidor manda al cliente datos periódicamente. La batería externa alimenta tanto la Raspberry como el dron proporcionando veinte minutos de autonomía, tiempo suficiente para tomar las distintas medidas. Un esquema completo del sistema que se ha diseñado se muestra en la figura 3.1.

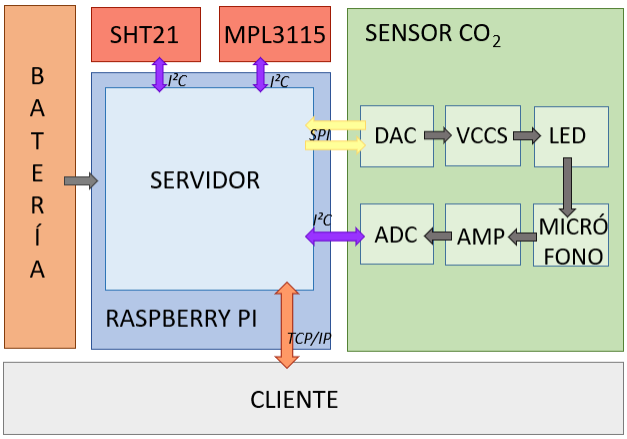


Figura 3.1: Esquema del sistema completo. Cliente y servidor se comunican mediante TCPI/IP. El servidor se comunica con el SHT21, MPL3115 y ADC mediante I^2C, mientras que con el DAC se comunica por SPI. El sensor de CO_2 está formado por: DAC, VCCS, LED, MEMS, AMP y ADC.

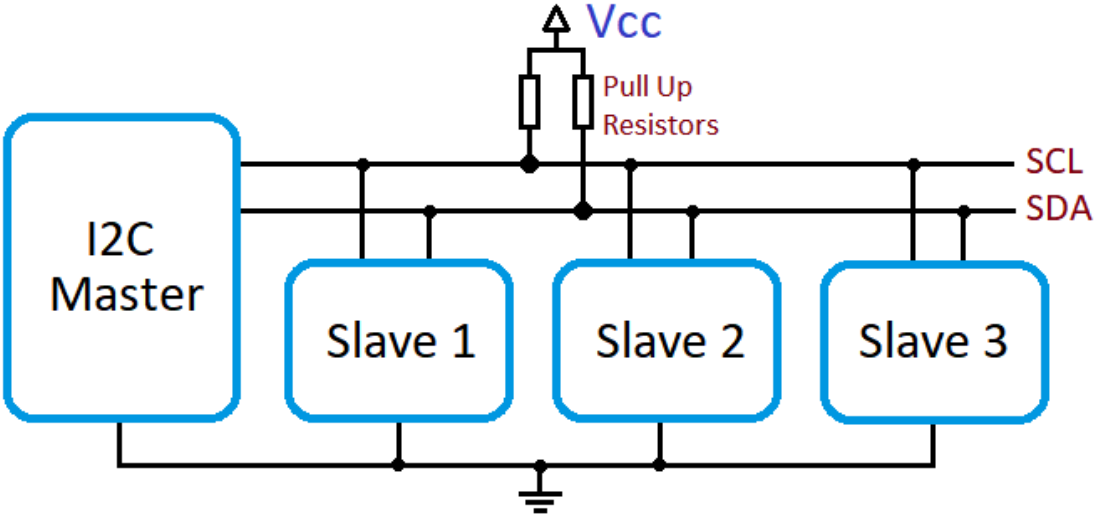
[1] Sensirion,Datasheet SHT21, web:http://www.farnell.com/datasheets/1780639.pdf [Fecha de acceso: 17-06-2019]

[2] Freescale Semiconductor, Inc., Datasheet MPL3115, web: https://www.cypress.com/file/50966/download [Fecha de acceso: 17-06-2019]

[3] Raspberry Pi Foundation, Raspberry Pi 3 Model B +, web: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf [Fecha de acceso: 17-06-2019]

3.1. Integración de microsensores de Temperatura, Humedad y Presión

El protocolo I^2C está basado en una topología maestro-esclavo, siendo la Raspberry Pi el maestro y los distintos sensores los esclavos. Lo cual significa que es la Raspberry la que tiene la iniciativa en la comunicación, generando una señal de reloj, necesaria para la sincronización entre todos los dispositivos, y controlando cuándo y a quién se transmiten datos. Esto se hace a través de dos líneas de comunicación: Serial Clock (SCL) y Serial Data (SDA) , correspondientes a las líneas de reloj y datos respectivamente. Cada esclavo es identificado de manera única por su dirección en el bus. De esta forma el protocolo garantiza la comunicación entre el maestro y cada uno de los esclavos de forma inequívoca. Las direcciones de memoria de cada uno de los sensores es la siguiente:
SHT21: 0x40
MPL3115: 0x60



.

Figura 3.2: Topología del protocolo I^2C. El maestro (master) se comunica con los demás sistemas mediante las líneas SCL y SDA de reloj y datos respectivamente. Ambas líneas son compartidas por cada uno de los esclavos (slaves) [15].

En el protocolo I^2C las transferencias de datos comienzan con una señal de START que se produce cuando la línea de datos SDA pasa de alto a bajo y SCL permanece en alto; y finaliza con una señal de STOP en la que la línea SDA pasa de bajo a alto y SCL permanece en alto. El formato de una tranferencia varía con respecto a cada sensor, exceptuando las condiciones de comienzo y de parada. Tal y como aparece en la figura 3.3, el primer byte se corresponde a la dirección en el bus del sensor o sistema con el que se quiera comunicar el maestro, seguido de un bit que indica lectura o escritura. Gracias a ese bit el protocolo I^2C utiliza un único bus de datos bidireccional. Un ejemplo de transferencia I^2C se muestra en la figura 3.3.

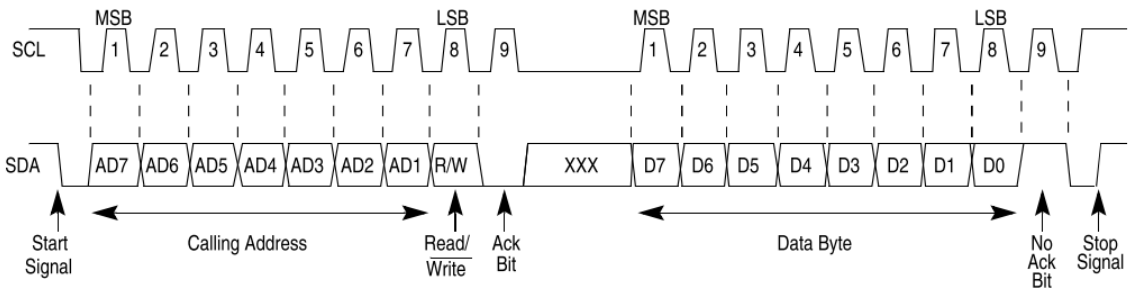


Figura 3.3: Ejemplo de transferencia I^2C. Empieza con la señal de START: la línea SDA pasa de alto a bajo mientras que SCL se mantiene en alto. A ello le sigue el primer byte de datos referido a la dirección del esclavo con el que se quiera comunicar el maestro. Dicha dirección es de 7 bits, y se le concatena un último bit que indica si la acción es lectura o escritura. A continuación se envían los diferentes bytes de datos (en este ejemplo es un único byte de datos). Y se finaliza la comunicación con la señal de STOP: la línea SDA pasa de bajo a alto mientras que SCL permanece en alto [16].

3.2. Diseño del sensor de CO_2

Como se ha mencionado en el Estado de Arte 2, no existen sensores de CO_2 de tamaño y peso lo suficientemente pequeño como para ser montados en un dron y midan dicho gas con buena resolución. Por ello se ha desarrollado el siguiente microsensor de CO_2:

Figura 3.4: En esta figura se muestra el LED (a), cápsula receptora (b) y cúpula (c) que comforman el sensor de CO_2

El sistema posee dos elementos básicos llamados emisor y receptor. El emisor es un LED cuya conducta está controlada por un DAC, mientras que el receptor es un micrófono que medirá la energía que hay en el sistema. Por otro lado, gracias al principio de atenuación de la luz según la Ley de Beer-Lambert 2.1 sabiendo la energía inicial generada y la resultante, se conocerá el CO_2 que hay en el espacio ambiente puesto que la energía que reciba el receptor va a ser menor que la emitida y conocida por el LED.

El DAC es controlado por el protocolo de comunicación SPI y genera una onda sinusoidal a una frecuencia de 100Hz y amplitud de 350mV. Por otra parte, el ADC receptor es controlado por I^2C y se encarga de regenerar la onda que modela la energía que hay en el sistema. Además, tal y como se

observa en la figura 3.4, la parte superior del sensor es una superficie elipsoidal en la cual el emisor y receptor se encuentran en cada uno de sus focos, aumentando de esta forma la capacidad senstiva del sistema.

El protocolo SPI, a diferencia del I^2C, utiliza 3 buses: Master Output System Imput (MOSI) , Master Input System Output (MISO) y SCLK; además de n líneas Slave Select (SS) , una por cada dispositivo SPI. Sigue también un esquema maestro-esclavo utlizando los buses MOSI y MISO para la transmisión de datos, siendo el primero para la comunicación desde la Raspberry Pi hacia el DAC y el segundo desde el DAC hacia la Raspberry Pi. Cada uno de ellos es, por tanto, unidireccional mientras que I^2C utiliza un único bus de datos bidireccional. La línea SCLK es necesaria para la sincronización entre ambos dispositivos. Por otro lado, las señales SS son las encargadas de seleccionar el sensor o sistema con el que se va a comunicar el maestro. En este caso, únicamente hará uso de SPI el DAC por lo que habrá una única señal SS.

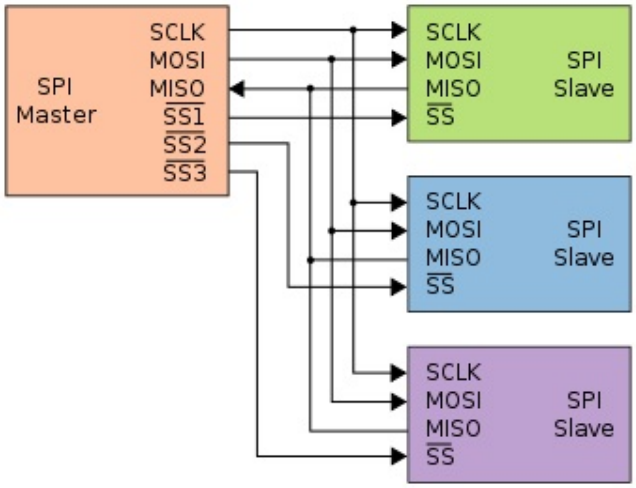


Figura 3.5: Ejemplo de topología del protocolo SPI que incluye el maestro y tres dispositivos. Formado por tres líneas de comunicación SCLK, MOSI y MISO, junto con tres señales SS correspondientes a cada sistema. Dichas señales los determina de manera única.

Para explicar cómo está formado el sensor y con el fin de que se entienda de manera sencilla se divide en lo que se considera que son sus partes fundamentales: emisor, receptor y cúpula. El esquema del sensor se muestra en la figura 3.6.

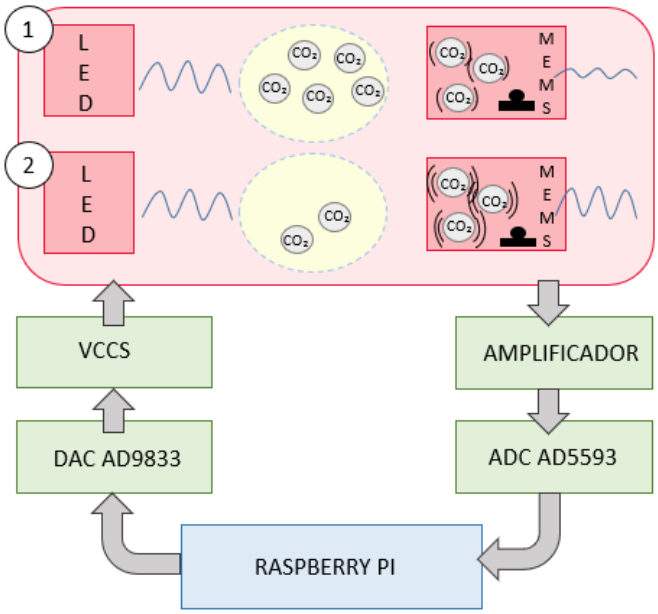


Figura 3.6: Esquema del sensor de CO_2. La Raspberry Pi se encarga de gestionar el sensor. El recpetor está formado por el DAC AD9833, un VCCS y el LED L13201. El emisor está formado por un micrófono MEMS, un amplificador y el ADC AD5593 que recoge los resultados y los comunica de nuevo a la Raspberry Pi.

Emisor

El emisor es el LED L13201 [4](Hamamatsu) controlado por el DAC AD9833 [5](Analog Devices) y que emite luz, es decir, fotones, proporcionales a una onda sinusoidal de amplitud 350mV y frecuencia 100Hz. Este LED trabaja en una longitud de onda alrededor de 4.3μm. La absorción de luz por parte del CO_2 es mayor en la franja 4.2μm - 4.4μm por lo que de haber CO_2 en el espacio ambiente el sensor lo detectará con mayor precisión que con un LED que emita fotones en otra longitud de onda. Además, en esta franja no hay aborción de luz por parte del Agua (H_2O), por lo que no habrán interferencias por humedad.

Entre el DAC y el LED hay un VCCS que genera una corriente proporcional al voltaje del DAC y de forma controlada. El VCCS empleado en este proyecto se muestra en la figura 3.7.

[4] Hamamatsu, Datasheet L13201 [Fecha de acceso: 17-06-2019]

[5] Analog Devices, Datasheet AD9833 [Fecha de acceso: 17-06-2019]

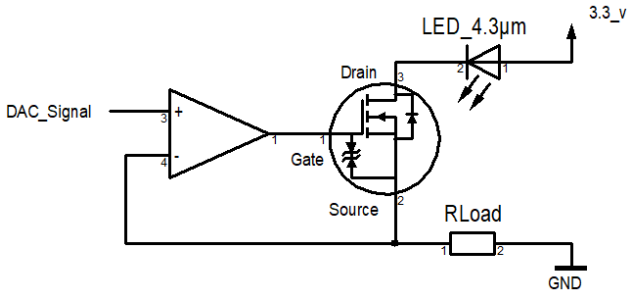


Figura 3.7: El VCCS está compuesto por un amplificador operacional que permite el control sobre la corriente con el fin de que no se queme el LED, y un transistor tipo Metal-Oxide-Semiconductor Field-effect Transistor (MOSFET) conectado al LED.

Receptor

El receptor es un sistema de mayor complejidad. Se trata de una cápsula hermética que contiene en su interior CO_2 y un micrófono MEMS ("SPU0410HR5H1", Knowles Electronics). Cuando los fotones atraviesan la cámara, calientan el interior de la cápsula, haciendo, por tanto, que las moléculas de CO_2 se muevan, variando la temperatura y presión. Aplicando el principio de variación de presión y temperatura de los gases ideales [17] donde P es la presión, V el volumen, n la concetación del gas, R la constante universal de los gases ideales y T la temperatura

$$PV = nRT \tag{3.1}$$

obtenemos una variación de presión en el interior, es decir, sonido, propocional a la intensidad de luz que esta recibiendo y, por consiguiente, a la onda enviada por el LED. El voltaje producido por el micrófono se amplifica y es recogido por el ADC AD5593 [6](Analog Devices) encargado de modular dicha onda para poder obtener la información necesaria de ella.

Cúpula

Con la finalidad de que el receptor captara un mayor número de fotones se ha utilizado una superficie elipsoidal revestida con una capa de oro de aproximadamente 200nm y se ha situado al emisor y al receptor en cada uno de los focos de la elipse como se ve en la figura 3.8. De esta forma, toda la luz que emite el emisor es recibida por el receptor. Adicionalmente, el oro es un material con una capacidad máxima de reflectancia a partir de las longitudes de onda de 1μm lo que hará que incremente aún más la capacidad sensitiva del sistema.

[6] Analog Devices, Datasheet AD5593(2018)

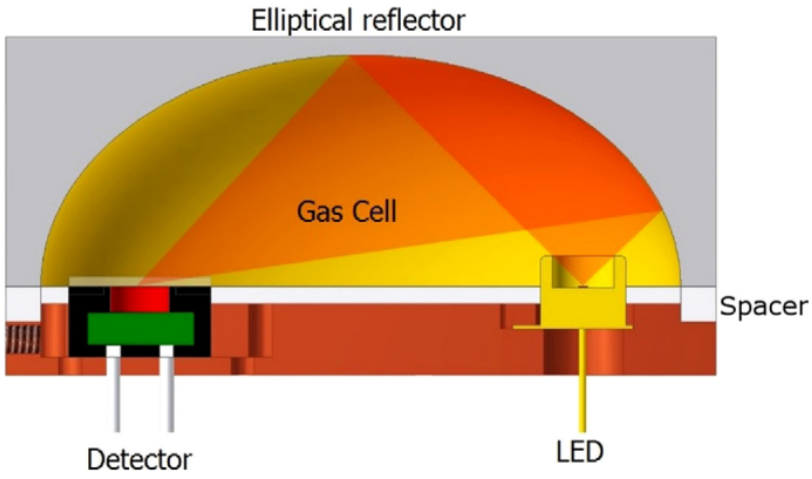


Figura 3.8: Cúpula del sensor de CO_2. El emisor y el receptor están en cada uno de los focos de la elipse por lo que la cantidad de fotones que reciba el receptor será máxima [18].

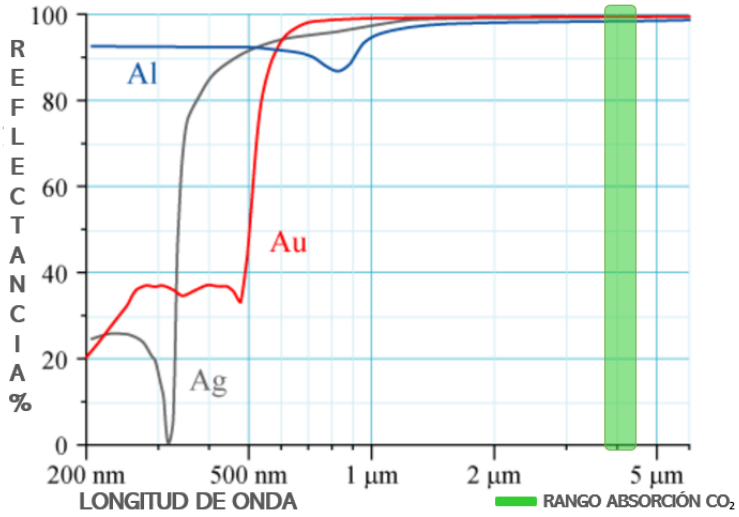


Figura 3.9: Reflectancia del Oro (Au), Plata (Ag) y Aluminio (Al) respecto a la longitud de onda. Por debajo de 500nm se comportan de forma diferente, siendo el Aluminio el que mejor resultados aporta. Sin embargo, a partir de 1μm los tres materiales tienen un 100 % de reflectancia.

Es decir, cuando haya CO_2 en el aire el número de fotones que llegue al receptor será menor debido a la capacidad de absorción de energía que tiene dicho gas y, por tanto, el incremento de temperatura en el interior de la cápsula será menor, al igual que la presión y, por consiguiente, el voltaje generado. De esta forma, sabiendo la frecuencia y amplitud tanto de la onda generada por el LED como de la reconstruida por el ADC, se conocerá el CO_2 que hay en el espacio ambiente.

3.3. Algortimos FFT y Goertzel

Una vez obtenidos los datos de voltaje del ADC AD5593 es necesario analizarlos para conocer la amplitud de la onda recibida y de esta forma saber el CO_2 que hay en el espacio ambiente. Para ello se ha implementado dos algoritmos diferentes: FFT [19] y Goertzel [20].

Fast Fourier Transform

El algoritmo FFT tiene como base la Transformada Discreta de Fourier (DFT) sin embargo, el hecho de que se le haya caracterizado como rápido (Fast) radica en la eliminación de cálculos repetitivos que se producen en la versión discreta. Para enseñar dicha redundancia se apunta la fórmula de la DFT para una serie de muestras x[n]:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-jkn\frac{2\pi}{N}} = \sum_{n=0}^{N-1} x[n] W_N^{kn} \tag{3.2}$$

siendo N el número total de muestras, n la n-ésima muestra original y k el k-ésimo término de la DFT. Además se tiene que:

$$W_N^{nk} = e^{-jkn\frac{2\pi}{N}} = \cos\left(\frac{2\pi}{N}\right) - j\,\mathrm{sen}\left(\frac{2\pi}{N}\right) \tag{3.3}$$

Se observa que el término W_N^{kn} es de carácter periódico y, por consiguiente, redundante. Por lo tanto, tenemos que la FFT descompone la señal en suma de senos y cosenos de diferentes frecuencias y amplitudes desfasadas en el tiempo.

Goertzel

El algoritmo Goertzel (código en el capítulo A) se basa también en la transformada de Fourier, sin embargo, es mucho más rápido que la FFT, utiliza menos operaciones matemáticas y requiere menos memoria [21]. Este algortimo se puede considerar un filtro paso-banda cuyo ancho es inversamente proporcional al número de muestras que se tengan para analizar. Se basa en las siguientes ecuaciones:

$$\nu[n] = 2\cos\left(2\pi\frac{k}{N}\right)\nu[n-1] - \nu[n-2] + x[n] \tag{3.4}$$

$$X[k] = \nu[N] - \nu[N-1]e^{-j\frac{2\pi}{N}} \tag{3.5}$$

donde n es la n-ésima muestra, $\nu[n]$ es el vector interno de memoria, k es el componente frecuencial, N el número total de muestras, x[n] las muestras de la señal y X[k] es el valor del k-ésimo término del espectro. Con el fin de eliminar el factor complejo de la ecuación 3.5, el valor de la potencia de la señal

es calculado de la siguiente forma:

$$P_k = |X[k]||X^*[k]| = \nu^2[N] + \nu^2[N-1] - 2\cos\left(2\pi\frac{k}{N}\right)\nu[N]\nu[N-1] \tag{3.6}$$

donde P_k es la potencia de la señal y X^* es el conjugado de X.

3.4. Interfaz cliente-servidor

Con el fin de crear una aplicación completa, se ha implementado un sistema cliente servidor, en el cual el cliente notifica al servidor cuando quiere comenzar y terminar de tomar medidas ya que el sistema se encontrará en un dron y tiene que ser manejado telemáticamente. Se trata de un sistema de comunicación cliente-servidor basado en el protocolo de comunicación TCP/IP. El cliente será una aplicación base mientras que el servidor se encontrará en la Raspberry.

Cuando arranca el dron se ejecuta el código del servidor, quedándose este en espera de peticiones de conexión. En el momento en el que el cliente se conecta y manda una petición al servidor de inicio de medidas, el servidor interactúa con los distintos sensores y envía los datos al cliente cada 5 segundos hasta que el cliente ordena la finalización de toma de medidas. Así mismo, cada vez que la aplicación cliente recibe un nuevo paquete de datos, los guarda en un fichero para así poderlos analizar posteriormente.

4

Resultados

Con el fin de garantizar el correcto funcionamiento del sensor de CO_2 se ha estudiado más a fondo el comportamiento tanto del emisor como del receptor. De esta forma se conocerán los parámetros adecuados de configuración de cada uno de ellos para así obtener su máximo rendimiento.

4.1. Evaluación del DAC AD9833

El LED L13201 es controlado por el DAC AD9833. A su vez, dicho DAC es manejado mediante el protocolo de comunicación SPI como se ha mencionado en la sección 3.2. Dicho DAC tiene como entrada, aparte de las líneas necesarias para la comunicación y las señales Common Collector Voltage (VCC) y Ground (GND) , una señal de reloj. A partir de ella, es capaz de generar cualquier tipo de onda. La señal de reloj es una onda cuadrada de 2.4MHz de frecuencia y 2.4V de amplitud. La frecuencia de corte del filtro es de 2.84MHz, es decir, que el ruido con frecuencia por encima de esta, será atenuado.

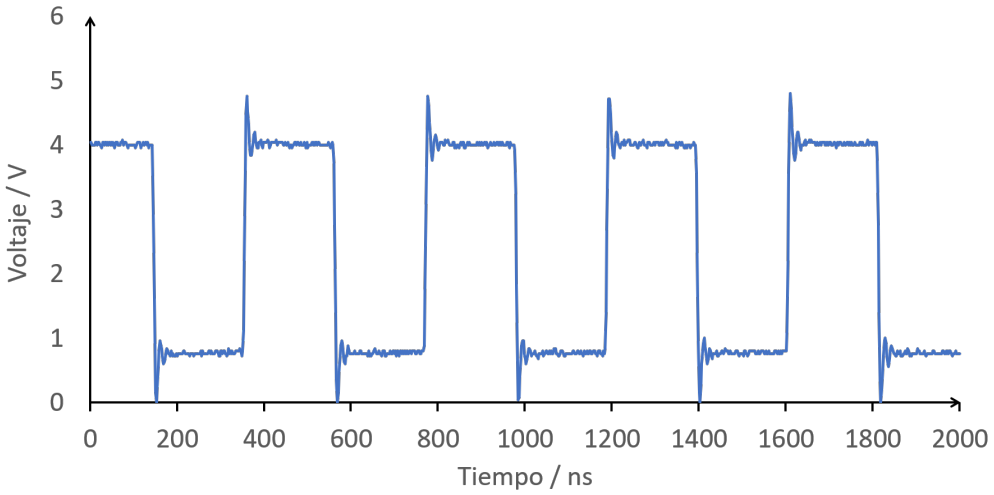


Figura 4.1: Onda cuadrada de 2.4MHz de frecuencia y 2.4V de amplitud generada por la Raspberry Pi 3 b + y utilizada como reloj para el DAC AD9833. El ruido de la señal es producido por la misma Raspberry.

Esta señal de reloj tiene ruido y eso afecta a la onda resultante. En un primer momento se aplicó el

filtro paso bajo de la figura 4.2(a) sobre ella, de modo que frecuencias más altas, causantes del ruido, se eliminaron. Sin embargo, de esta forma, el DAC no detectaba la señal del reloj, por lo que finalmente se decidió aplicar el filtro paso bajo de la figura 4.2(b) a la señal de salida. Este nuevo filtro tiene como frecuencia de corte 159KHz por lo que las componentes de la señal que tengan frecuencia mayor a dicha, se atenuarán, es decir, disminuirán de amplitud. Es por ello por lo que en las gráficas 4.3(d) y 4.3(e) la amplitud es menor que en la señal original.

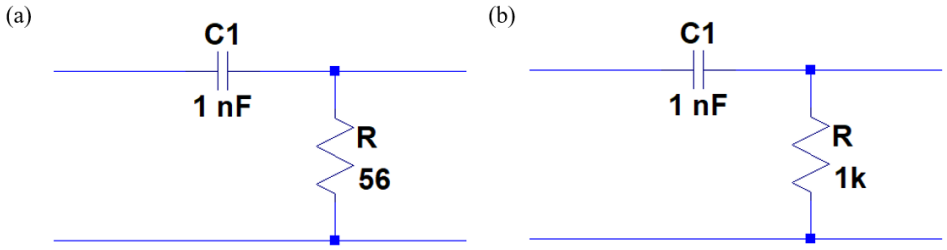


Figura 4.2: La figura (a) muestra el filtro paso bajo aplicado a la señal de reloj de la figura 4.1. Es un filtro pasivo compuesto por una capacidad de 1nF y una resistencia de 56ω. Su frecuencia de corte es 2.84MHz. En la figura (b) se muestra el filtro paso bajo aplicado a la onda generada por el DAC AD9833. Es un filtro pasivo compuesto por una capacidad de 1nF y una resistencia de 1Kω. Su frecuencia de corte es 159KHz.

La reducción de ruido es notable. En la figura 4.3 se muestra la onda generada por el DAC a frecuencias 1000Hz, 100KHz y 1MHz con y sin el filtro de la figura 4.2(b). La calidad de la onda final es peor cuanto mayor es su frecuencia y esto es debido a que la señal que utiliza el DAC para generar la onda es, efectivamente, el reloj. Dicho reloj funciona a una frecuencia de 2.4MHz, es decir, para generar una onda de, digamos, 10KHz, el DAC utliza 240 Puntos por Periodo (PP) : 2400000/10000. Según el teorema de Nyquist [22], el mínimo número de puntos necesario por periodo para definir la frecuencia de una onda es 2. Lo que hace que, en este ejemplo, 240 pp defina una señal de alta calidad. Por lo tanto, cuanto mayor sea la frecuencia de la señal de salida, peor definida estará. Para determinar la frecuencia límite se utiliza nuevamente Nyquist. Con menos de 2 pp, esto es, a una frecuencia mayor a 1MHz ya no sería posible conocer la frecuencia de la onda. Un claro ejemplo de este comportamiento se puede apreciar en la figura 4.3(e).

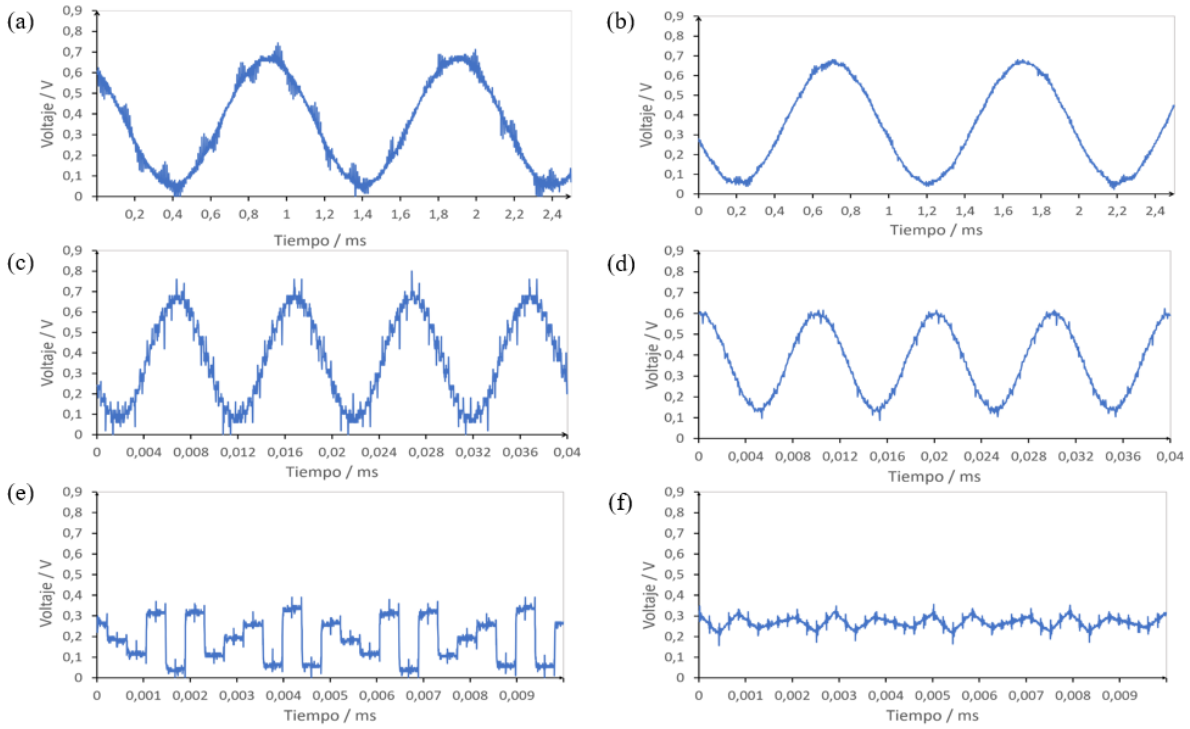


Figura 4.3: Comparación de ondas de 1KHz, 100KHz y 1MHz con y sin el filtro 4.2(b). El DAC es capaz de producir ondas de 1KHz y 100KHz de calidad, sin embargo no es capaz de llegar a una onda de 1MHz puesto que sólo dispone de 2-3 pp para generarla. Según Nyquist son necesarios 25 pp para generar una onda con buena resolución. Con el filtro 4.2(b) se reduce el ruido en un 90 %.

4.2. Evaluación del ADC AD5593

En la sección anterior se ha explicado y estudiado la parte del emisor del sensor de CO_2. Se muestra ahora en qué consiste el receptor con mayor detalle.

Las moléculas de CO_2 se calientan y enfrían de forma proporcional a la energía que reciben, es decir, la emitida por el LED (onda sinusoidal de 100Hz de frecuencia y 350mV de amplitud). Por la Ley de los gases ideales 3.1 esto produce una variación de la presión en el interior de la cápsula proporcional también a dicha onda y que mediante un micrófono MEMS produce un voltaje. Después de una fase de amplificación, ese voltaje generado es recogido por el ADC AD5593 que se encarga de recomponer la onda. La similitud entre la onda que recompone el ADC y la real está relacionada directamente con la cantidad de medidas o muestras por periodo que se realicen.

Se ha realizado un estudio sobre cómo influye en la frecuencia y amplitud de la onda digitalizada la cantidad de muestras por periodo que se toman sobre esta. Con el objetivo de obtener dicha información se han utilizado los dos algoritmos exaplicados en la sección 3.3: FFT y Goertzel.

Tomando una medida por periodo, es decir, con una frecuencia de muestreo igual a la frecuencia

de la onda inicial, el resulado es el siguiente: la onda resultante es tan diferente de la inicial que las componentes frecuenciales que se obtienen de dichos algoritmos no coinciden con la frecuencia de la onda real. El resultado que proporcional el Goertzel sobre la componente frecuencial de 100Hz es 0. En la gráfica 4.4(b) se muestran los resultados de aplicar el algoritmo FFT.

A continuación se toman medidas con el criterio Nyquist, dos veces por periodo, y se obtiene el siguiente resultado mostrado en la figura 4.4(c). Aplicando el algoritmo Goertzel a estos datos, se obtiene para la componente frecuencial de 100Hz una amplitud de: 686,008556 counts, equivalente a 0.4187V. En la gráfica 4.4(d) se muestran los resultados de aplicar el algoritmo FFT. Ambos algoritmos detectan la componente frecuencial de 100Hz.

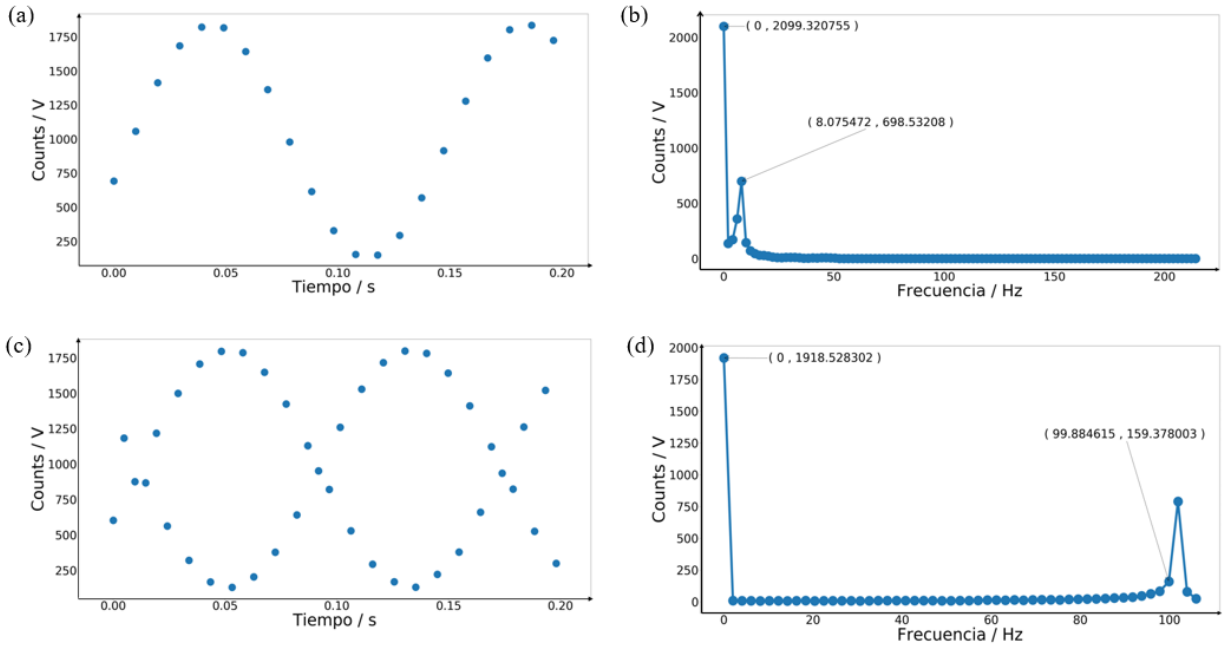


Figura 4.4: Onda de 100Hz reconstruida a partir de una frencuencia de muestreo de 107Hz y 212Hz. En las figuras (a) y (c) se muestra los datos obtenidos del ADC. En las figuras (b) Y (d) se muestran los resultados de aplicar el algoritmo FFT a los datos anteriores

El teorema de Nyquist afirma que el mínimo número de muestras por ciclo para obtener algo de información de la señal inicial es dos. No obstante esta frecuencia de muestreo vale únicamente en señales con muchas repeticiones y poca variación, es decir, con poco ruido, o en un tal caso, con ruido uniforme. Para este proyecto se genera ruido por lo que el criterio de Nyquist de 2 pp no da buenos resultados.

Se pueba ahora con una frecuencia de muestreo de 825Hz es decir, ocho veces más rapida que la inicial, por lo que se toman ocho medidas por periodo. El resultado de aplicar el algoritmo Goertzel para obtener la componente frecuencial de 100Hz: 570.98701892 counts, equivalente a 0.3485V; mientras que la FFT da como resultado 515.088244 counts, equivalente a 0.3144V. Muestreando a 825Hz sobre una onda de 100Hz se obtiene información de la frecuencia de la onda. Sin embargo puede suceder

que no haya ningún punto de medida que coincida con el pico de la onda, por lo que el valor de la amplitud aún no se puede obtener con toda garantía.

Por último, se ha decidido muestrear 35 veces por periodo. De esta forma, se obtienen los valores de amplitud y frecuencia de la onda y la diferencia entre la recreada y la inicial es menor que un 0.05 %. El esultado de aplicar el algoritmo Goertzel para obtener la componente frecuencial de 100Hz es 795.97992486 counts, equivalente a 0.4858V; mientras que para la FFT es 770.480806 counts, 0.4703V.

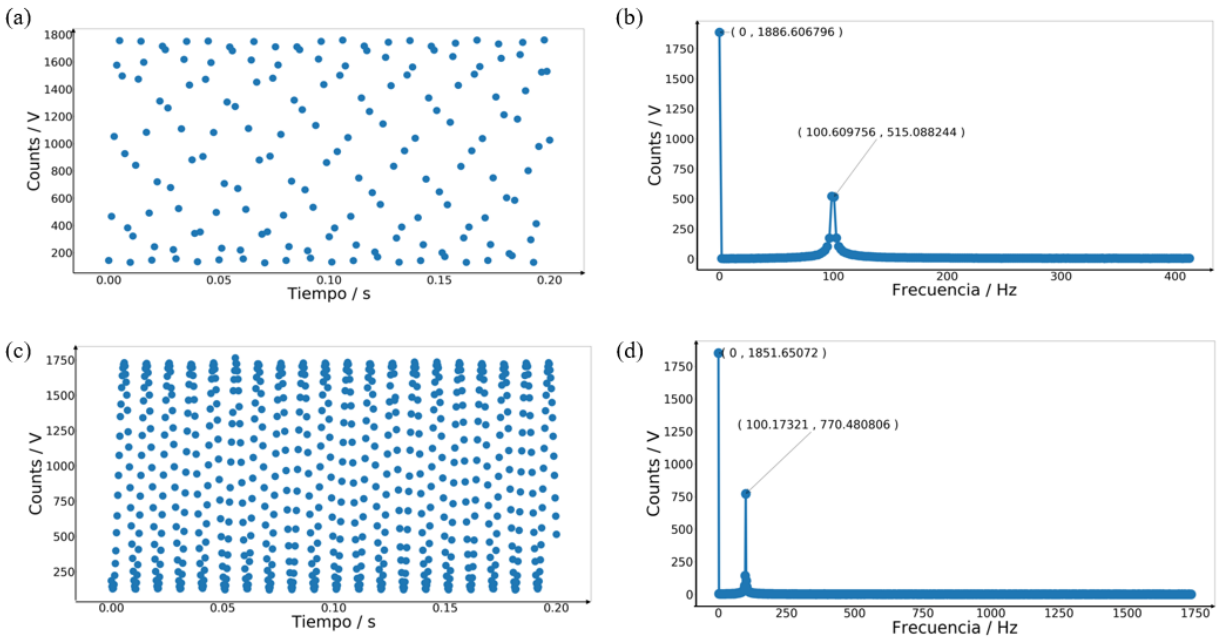


Figura 4.5: Onda de 100Hz reconstruida a partir de una frencuencia de muestreo de 825Hz y 3470Hz. En las figuras (a) y (c) se muestra los datos obtenidos del ADC. En las figuras (b) Y (d) se muestran los resultados de aplicar el algoritmo FFT a los datsos anteriores.

Ambos valores, sobre todo el de la amplitud, es fundamental para el cálculo posterior de la cantidad de CO_2 que hay en el aire, ya que, como se ha explicado anteriormente, el CO_2 absorbe energía y, sabiendo la diferencia de amplitud entre la onda emitida y la recibida, se conocerá el valor de dicha cantidad.

El factor de conversión entre counts y voltios es $\frac{2,5}{4096}$, siendo 2.5V el voltaje de referencia del ADC y 12 bits su resolución ($2^{12} = 4096$).

Captura de la onda con el fotodetector

Con el fin de comprobar que la onda sinusoidal generada con el DAC es equivalente a la onda de fotones que genera el LED, se ha analizado la luz del LED con el fotodetector de infrarrojos PDA20H (1.5μm - 4.8μm, Thorlabc). Los datos han sido recogidos con el osciloscopio y esta es la relación entre

ambas señales:

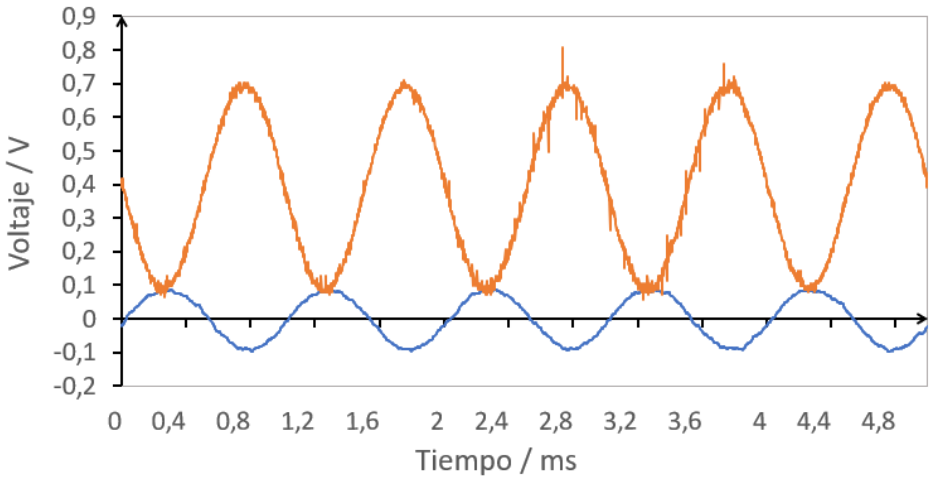


Figura 4.6: En naranja se muestra una onda de 1000Hz generada por el DAC AD9833 y en azul dicha onda emitida por el LED L13201. Ambas son proporcionales, estando la onda emitida por el LED desfasada con respecto a la señal del DAC.

Ambas ondas tienen la misma frecuencia y mantienen una relación en la amplitud, es decir, el LED envía una onda de fotones proporcional a la recibida por el DAC. En la figura 4.6 se observa, además, que la señal azul tiene un desfase de 0.5ms debido a la electrónica del propio fotodetector.

4.3. Medición de CO_2

En primer lugar se muestra gráficamente en la figura 4.7 la onda generada por el DAC AD9833 y que emite el LED, frente a la que recibe el ADC AD5593. Se observa la relación entre la onda recibida por el ADC y la emitida por el LED. Ambas son proporcionales, lo que permite obtener la cantidad de luz absorbida por el gas y, por tanto, la concentración de este. Se aprecia también el ruido que presenta el sistema y que provoca la deformación de la señal. Cabe señalar el desfase de 3ms de la señal recibida frente a la emitida.

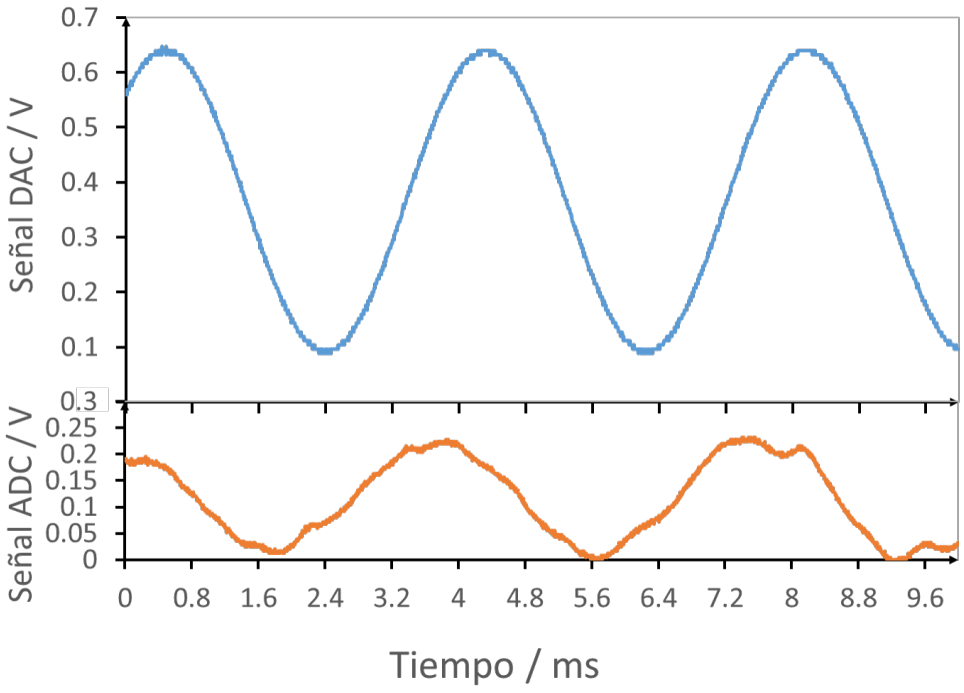


Figura 4.7: Onda generada por el DAC AD9833 frente a la señal recibida por el ADC AD5593. Se observa el ruido que presenta el sistema y el desfase entre ambas ondas, pues la señal recogida por el ADC es debido a la generada por el DAC.

Se ha realizado una medición de dióxido de carbono para comprobar el correcto funcionamiento del sensor. Para ello se ha tomado medidas durante unos minutos y en dos instantes se ha exhalado aire a unos centímetros del sensor. Los resultados se muestran en la gráfica 4.8.

En los primeros instantes los resultados de la FFT y Goertzel oscilan entre valores de 160 y 200 counts. En el instante 70 segundos hay un mínimo en la gráfica en el que el resultado de los algoritmos cae de 150 counts a 40. Corresponde al primer momento en el que se exhala a unos centímetros del sistema de mediciones expulsando, por tanto, CO_2. Las moléculas de CO_2 provocan que los fotones emitidos por el LED sean recibidos con menos intensidad en el repector debido a la capacidad de absorción de luz por parte de este gas explicada en la sección 3.2. Esto hace que la onda generada por el micrófono y el ADC sea de una aplitud menor, en este caso, 40 counts. Unos segundos después el sistema recoge otro aumento de CO_2. Esta vez se sopla en el sensor y es por ello por lo que alcanza

de nuevo un valor de 50 counts. A diferencia de la ocasión anterior, no se exhala con tanta intensidad (como si se quisiera hacer vaho) sino que se sopla, y es por ello por lo que la humedad no presenta el mismo pico que en el instante 70. Para obtener los datos finales de concentración de dióxido de carbono es necesaria la calibración del sensor, la cual no se ha podido efectuar por la inhabilitación del laboratorio.

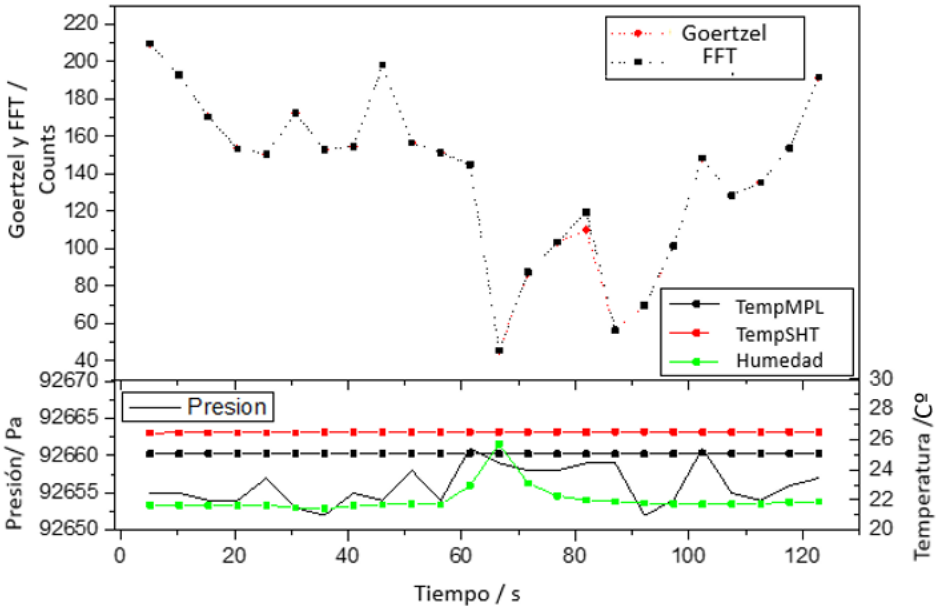


Figura 4.8: Datos recogidos tras exhalar aire a una distancia corta del sistema de mediciones. Recoge los datos de temperatura, humedad, presión, FFT y Goertzel.

5

Conclusiones

Se ha diseñado y desarrollado un sistema de bajo coste, peso y dimensión capaz de tomar medidas de temperatura, humedad, presión y concentración de CO_2 bajo una interfaz cliente-servidor, controlado por una Raspberry Pi 3 b + y con las condiciones necesarias para poder ser integrado en un dron. Todo ello ha sido posible gracias a la elección de dos sensores comerciales de bajo coste y dimensiones (SHT21 y MPL3115) encargados de medir temperatura, humedad y presión; y el diseño de un sensor de dióxido de carbono NDIR basado en detección fotoacústica y en el principio de absorción de los gases de Beer-Lambert (2.1). El control de los sensores de temperatura, humedad y presión se ha hecho mediante el protocolo de comunicación I^2C, mientras que la comunicación cliente-servidor por TCP/IP.

Por otro lado, la parte emisora del sensor de CO_2 ha sido manejada con el DAC AD9833 mediante el protocolo de comunicación SPI. El estudio realizado sobre dicho DAC concluye que la frecuencia límite que éste puede proporcionar sin excesivo ruido es de 100KHz, teniendo como entrada un reloj de 2.4MHz. Dicho resultado concuerda con el Teorema de Nyquist que determina que son necesarios 25 puntos por periodo para modelar una onda. El receptor del sensor es el ADC AD5593, manejado también por la Raspberry mediante I^2C y es usado para digitalizar la onda recibida por el micrófono. La frecuencia límite a la que es capaz de muestrear este ADC es 6KHz y, teniendo en cuenta de nuevo Nyquist, se ha elegido que la onda que emita el LED sea de 100Hz. De esta forma el ADC es capaz de digitalizar la onda que reciba de forma exacta para posteriormente obtener el dato de la amplitud mediante el algoritmo Goertzel, óptimo temporalmente frente a la FFT puesto que analiza únicamente una componente frecuencial o, en su defecto, un rango; frente a la FFT que aplica el algoritmo a todo el espectro frecuencial.

La amplitud de la onda recogida por la parte receptora del sensor juega un papel fundamental a la hora de determinar la concentración de CO_2 pues, siendo conocida la amplitud de la onda que emite el LED y, adicionalmente, la amplitud de la onda recibida, se conocerá la diferencia de energía y, por tanto, la absorbida por el gas. Aportando de esta forma un dato concreto sobre la concentración de éste en el espacio ambiente. Así mismo, tras realizar diferentes mediciones de dióxido de carbono se concluye que, con este diseño de sensor, no influye ni la temperatura ni la humedad a la hora de determinar su concentración en el espacio ambiente.

En un futuro y, con el fin de que este sistema sea integrado en un dron y envíe datos a una aplicación base de forma eficaz, se modificará de protocolo de comunicación TCP/IP bajo el cual está implementada la estructura cliente-servidor debido a que cuando el dron llegue a altas posiciones la señal wifi se perderá. Se implementará esta estructura mediante otro protocolo de comunicación basado en radio frecuencias y que abarque largas distancias como Long-Range (LoRa) . Así mismo, se utilizará un microcontrolador PSOC en lugar de la Raspberry Pi con el objetivo de tener un control más exhaustivo sobre los relojes del sistema y los diferentes dospositivos. A razón de no estar aún habilitado el laboratorio, no ha sido posible realizar la calibración correspondiente al sensor de CO_2 por lo que no se han podido obtener datos finales de concentración de CO_2 en partes por millón (ppm) .

Bibliografía

[1] C. Tarnocai, J. G. Canadell, E. A. G. Schuur, P. Kuhry, G. Mazhitova, and S. Zimov, "Soil organic carbon pools in the northern circumpolar permafrost region," *Global Biogeochemical Cycles*, vol. 23, no. 2, 2009.

[2] C. Mota, C. Alcaraz, Iglesias Maria, B. Martínez, and C. Micaela, "Investigación sobre la absorción de CO2 por los cultivos más representativos," *Consejo superior de Investigaciones Cientificas*, vol. 1, p. 43, 2010.

[3] D. Chaudhary, S. Nayse, and L. Waghmare, "Application of Wireless Sensor Networks for Greenhouse Parameter Control in Precision Agriculture," *International Journal of Wireless & Mobile Networks*, vol. 3, no. 1, pp. 140–149, 2011.

[4] P. R. N. Childs, J. R. Greenwood, and C. A. Long, "Review of temperature measurement," *Review of Scientific Instruments*, vol. 71, no. 8, pp. 2959–2978, 2000.

[5] Z. Chen and C. Lu, "Humidity Sensors: A Review of Materials and Mechanisms," *Sensor Letters*, vol. 3, no. 4, pp. 274–295, 2005.

[6] R.-H. Ma, Y.-H. Wang, and C.-Y. Lee, "Wireless Remote Weather Monitoring System Based on MEMS Technologies," *Sensors*, vol. 11, no. 3, pp. 2715–2727, 2011.

[7] E. J. Calvo and M. Otero, "Chemical sensors," *Piezoelectric Transducers and Applications*, vol. 70, no. 12, pp. 241–257, 2008.

[8] W. Schmidt, *Optische Spektroskopie*. D-69451 Weinheim, Germany: Wiley-VCH Verlag GmbH, 2000.

[9] L. Scholz, A. Ortiz Perez, B. Bierer, P. Eaksen, J. Wöllenstein, and S. Palzer, "Carbon Dioxide Sensor for Mobile Devices," *IEEE Sensors*, vol. 07, no. 9, pp. 6–8, 2016.

[10] T. Yasuda, S. Yonemura, and A. Tani, "Comparison of the Characteristics of Small Commercial NDIR CO2 Sensor Models and Development of a Portable CO2 Measurement Device," *Sensors*, vol. 12, no. 3, pp. 3641–3655, 2012.

[11] T. V. Dinh, I. Y. Choi, Y. S. Son, and J. C. Kim, "A review on non-dispersive infrared gas sensors: Improvement of sensor detection limit and interference correction," *Sensors and Actuators, B: Chemical*, vol. 231, pp. 529–538, 2016.

[12] J. Hodgkinson, R. Smith, W. O. Ho, J. R. Saffell, and R. P. Tatam, "Non-dispersive infra-red (NDIR) measurement of carbon dioxide at 4.2 μm in a compact and optically efficient sensor," *Sensors and Actuators, B: Chemical*, vol. 186, pp. 580–588, 2013.

[13] Philips Semiconductors, "UM10204 I2C - bus specification and user manual Rev. 6-4 April 2014 User manual Document information Info Content," 2014.

[14] Texas Instruments, "KeyStone Architecture Serial Peripheral Interface (SPI) User Guide," *Texas Instruments*, no. November, p. 2, 2010.

[15] Sitio web VHDL IMPLEMENTATION I^{2}C, [Fecha de acceso: 10-06-2019].

[16] F. Semiconductor, "I 2 C Precision Altimeter MPL3115A2," pp. 1–37, 2011.

[17] O. Marti, *Vorlesungsskript Grundlagen der Physik II für Physiker, Wirtschaftsphysiker und Lehramtskandidaten*. 2007.

[18] V. Wittstock, L. Scholz, B. Bierer, A. O. Perez, J. Wöllenstein, and S. Palzer, "Design of a LED-based sensor for monitoring the lower explosion limit of methane," *Sensors and Actuators, B: Chemical*, vol. 247, pp. 930–939, 2017.

[19] J. Alexander, C. O. Francisco, A. Medina, and A. J. Andrés, "Del Análisis de Fourier a las Wavelets," *Scientia et Technica*, vol. XIII, no. 34, pp. 151–156, 2007.

[20] T. G. Algorithm, "The Goertzel Algorithm," *Embedded Systems programming*, no. September, 2002.

[21] H. Bruhns, A. Marianovich, and M. Wolff, "Photoacoustic Spectroscopy Using a MEMS Microphone with Inter-IC Sound Digital Output," *International Journal of Thermophysics*, vol. 35, no. 12, pp. 2292–2301, 2014.

[22] P. Boersma, "Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound ," *Institute of Phonetic Sciences*, vol. 17, pp. 97–110, 1993.

Acrónimos

ADC Analog-to-Digital Conversion.

CO_2 dióxido de carbono.

DAC Digital-to-Analog Conversion.

DFT Transformada Discreta de Fourier.

FFT Fast Fourier Transform.

GND Ground.

I^2C Inter-Integrated Circuit.

IP Internet Protocol.

IUPAC International Union of Pure and Applied Chemistry.

LED Light-emitting diode.

LoRa Long-Range.

MEMS Microelectromechanical System.

MISO Master Input System Output.

MOSFET Metal-Oxide-Semiconductor Field-effect Transistor.

MOSI Master Output System Imput.

NDIR Non-dispersive infrared spectroscopy.

PP Puntos por Periodo.

ppm partes por millón.

RTD Resistencia Temperatura Detector.

SCL Serial Clock.

SDA Serial Data.

SPI Serial Peripheral Interface.

SS Slave Select.

TCP Transmision Control Protocol.

VCC Common Collector Voltage.

VCCS Voltage Controlled Current Source.

Apéndices

A

Código algoritmo Goertzel

Código A.1: En esta figura se presenta el código correspondiente al algoritmo Goertzel.

```
1  def goertzel(samples, sample_rate, f_start, f_end):
2
3      window_size = len(samples)
4      f_step = int(sample_rate / float(window_size)) # JLD: in Hz
5      k_start = int(math.ceil(f_start / f_step))
6      k_end = int(math.floor(f_end / f_step))
7
8      if k_end > window_size -1: raise ValueError('frequency out of range %s' % k_end)
9
10     n_range = range(0, window_size)
11     freqs = []
12     results = []
13     f_step_normalized = 1./window_size # JLD: step in normalized frequency
14
15     for k in range(k_start, k_end + 1):
16         # Bin frequency and coefficients for the computation
17         f = k *f_step_normalized # JLD: here you need the normalized frequency
18         w_real = 2.0 *math.cos(2.0 *math.pi *f)
19         w_imag = math.sin(2.0 *math.pi *f)
20
21         d1, d2 = 0.0, 0.0
22         for n in n_range:
23             y = samples[n] + w_real *d1 -d2
24             d2, d1 = d1, y
25
26         results.append((
27             0.5 *w_real *d1 -d2, w_imag *d1,
28             d2**2 + d1**2 -w_real *d1 *d2) # removed factor 2: already in w_real!
29         )
30         freqs.append(f *sample_rate)
31     return freqs, results
```

B

Código del servidor y cliente

Código B.1: En esta figura se presenta el código correspondiente a la aplicación servidor. Primera parte.

```
1  def read_file():
2      file = open("datos.txt", 'r')
3      content = file.readlines()
4      data = [ float(x.rstrip('\n')) for x in content]
5      file.close()
6      return data
7
8
9  def start_measure(sht,mpl,sc):
10     '''
11     Toma mediciones de los sensores a frecuencia FREQSENS hasta llegar a tiempo
12     :param mpl: sensor mpl3115, barómetro
13     :param sht: sensor sht21, sensor de temperatura y humedad
14     :return dataSens: diccionario indexado por el tiempo con las mediciones de ambos sensores como valor
15     '''
16     #print(threading.currentThread().getName(), 'Starting')
17
18     global medidas
19     global socket_cliente
20     while medidas!=2:
21         dataSens={}
22         out = mpl.read_data()
23         temp = sht.read_temperature()
24         hum = sht.read_humidity()
25         t0 = time.time()
26         command_line = 'sudo ./Pckg/SENSORS/ad5593_2'
27         args = shlex.split(command_line)
28         subprocess.call(args)
29         time.sleep(2)
30         dataADC = read_file()
31         tdiff = dataADC[0] #la primera fila del fichero es lo que ha tardado en medir
32         sample_rate_c = int((len(dataADC)-1)/tdiff)
33         fft_size_c = int(sample_rate_c/F_STEP)
```

Código B.2: En esta figura se presenta el código correspondiente a la aplicación servidor. Segunda parte.

```
34
35          av_fft=0
36          av_g=0
37          for i in range(5):
38
39              yc = dataADC[i*fft_size_c+1:(i+1)*fft_size_c+1]
40              result_fft = f.filter_fft(yc,fft_size_c,sample_rate_c)
41              freqs_c, result_goertzel_c = f.filter_goertzel(yc, fft_size_c, sample_rate_c, FREQ_DAC,50,
                    True)
42
43              #hallamos datos de componente continua y frecuencia de la fft
44              comp_cont_fft=max(result_fft)
45              i_comp_cont_fft,=np.where(result_fft==comp_cont_fft)
46              copy_result_fft=result_fft.copy()
47              copy_result_fft= np.delete(copy_result_fft,i_comp_cont_fft)
48              max_fft = max(copy_result_fft)
49              av_fft = av_fft + max_fft
50
51              #hallamos datos de componente continua y frecuencia del goertzel
52
53              comp_cont_goert=max(result_goertzel_c)
54              av_g= av_g + comp_cont_goert
55
56          av_fft = av_fft/5
57          av_g = av_g/5
58
59          dataSens[t0]={'pressure':out['pressure'],'temperature':out['temp'],'tempSht':temp,'humedad':hum,
                'result_fft':av_fft, 'result_goertzel':av_g}
60
61          tf=time.time()
62          time.sleep(TIEMPO_MEDICIONES-(tf-t0))
63          socket_cliente.send(str(dataSens))
64          writer.writerows([[str(datetime.now()),str(out['pressure']), str(out['temp']),
                str(temp),str(hum),str(av_fft),str(av_g)]])
65
66
67      return dataSens #realmente ahora da igual que lo devuelva o no
68
69
70              #HILO MAIN 1
71
72  def decision(sc):
73      global medidas
74      while True:
75          msg = sc.recv(1024)
76          if msg == '1':
77              lock.acquire()
78              medidas = 1
79              lock.release()
```

Código B.3: En esta figura se presenta el código correspondiente a la aplicación servidor. Tercera parte.

```
80              if msg == '2':
81                  lock.acquire()
82                  medidas = 2
83                  lock.release()
84                  break
85
86
87              #HILO MAIN 2
88
89      def program(sc,sht21, mpl):
90          global medidas
91          while True:
92              if medidas ==1:
93                  ''' SE PONE EN FUNCIONAMIENTO EL PROGRAMA DE MEDIDAS, EL OBJETIVO ES
                        QUE MANDE DATOS CADA 5 SEGUNDOS AL SERVIDOR'''
94
95                  start_measure(sht21, mpl, sc)
96
97
98
99              if medidas ==2:
100                 print("Dejamos_de_mandar_datos")
101                 #writer.writerows(csvData)
102                 sc.close()
103                 csvFile.close()
104                 break
105
106
107
108     #PARTE PRINCIPAL DEL PROGRAMA COMPLETO CLIENTE.PY
109
110
111     socket_cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
112     socket_cliente.connect(("localhost", 9999))
113     #I2C device 1
114     sht21= sht21.SHT21(4)
115     mpl = m.MPL3115(4)
116     mpl.config(False,102200) # solo lee la presion
117                     #TODOS VAN A IR POR EL MISMO BUS PERO DE FORMA SECUENCIAL:
                            PRIMERO MEDIDAS DEL ADC Y LUEGO DE SHT Y MPL
118     #SPI
119     dac = dac.AD9833(10,11,8,2500000)
120     dac.set_freq(FREQ_DAC)
121
122     dec = threading.Thread(name='decision', target=decision, args= (socket_cliente,))
123     main = threading.Thread(name='main_program', target=program, args= (socket_cliente,sht21, mpl))
124     dec.start()
125     main.start()
126     dec.join()
127     main.join()
128
129     print ("Adios")
130     socket_cliente.close()
```

Código B.4: En esta figura se presenta el código correspondiente a la aplicación cliente. Primera parte.

```
1  import socket
2  import threading
3
4  lock = threading.RLock()
5  medidas = 0
6
7  def guardar(file, dato):
8      file.write(str(dato)+'\n')
9
10 def decision(sc):
11     #print(threading.currentThread().getName(), 'Starting')
12     print("Pulse 1 para empezar a tomar mediciones")
13     print("Pulse 2 para terminar de tomar mediciones")
14     global medidas
15     while True:
16
17         d = input(">>")
18         if d == 1:
19             lock.acquire()
20             medidas = 1
21             lock.release()
22             sc.send(str(d).encode('utf-8'))
23         if d == 2:
24             lock.acquire()
25             medidas =2
26             lock.release()
27             sc.send(str(d).encode('utf-8'))
28             break
29     print(threading.currentThread().getName(), 'Saliendo')
30     return medidas
31
32 def program(sc,s):
33     global medidas
34     #print(threading.currentThread().getName(), 'Starting')
35     file = open("info.txt", 'w')
36     while True:
37
38         if medidas == 1:
39             recibido = sc.recv(1024)
40             print ("Recibido: ", recibido)
41             guardar(file,recibido)
42         if medidas == 2:
43             print("Dejamos de tomar medidas")
44
45             sc.close()
46             s.close()
```

Código B.5: En esta figura se presenta el código correspondiente a la aplicación cliente. Segunda parte.

```
47                 break
48         print(threading.currentThread().getName(), 'Saliendo')
49         file.close()
50
51             # PARTE PRINCIPAL DEL PROGRAMA COMPLETO CLIENT.PY
52
53 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
54 s.bind(("127.0.0.1", 9999))
55 s.listen(5)
56
57 print ("Servidor de Chat\n")
58 sc, addr=s.accept()
59 print ("Cliente conectado desde: ", addr)
60 dec = threading.Thread(name='decision', target=decision, args= (sc,))
61 main = threading.Thread(name='main_program', target=program, args= (sc,s))
62 dec.start()
63 main.start()
64 dec.join()
65 main.join()
66
67 sc.close()
68 s.close()
```

C

Librerías implementadas

C.1. Librería para la configuración y gestión del ADC AD5593

Código C.1: En esta figura se muestra la librería implementada para la configuración y gestión del ADC AD5593. Primera parte.

```
1  class ADC:
2
3      _I2C_ADDRESS_ESCRITURA = 0x20
4      _I2C_ADDRESS_LECTURA = 0x21
5      _STATUS_BITS_MASK = 0xFFFC
6      _WAIT_TIME= 0.050
7      _BIT_MASK= 0x01
8      _BIT_REP=0X02
9      _BIT_BUFFER = 0x20
10     _BIT_2_VREF =0X20
11     _BIT_PD=0x02
12     _MASK_D15=0x80
13     _MASK_MSB= 0x0F
14     _CONVERSION= 2.5/4096
15     I2C_SLAVE = 0x0703
16     I2C_ADDR = 0x10
17
18     PB= {'adc_readback':0x40, 'gpio_readback':0x60, 'register_readback':0x70,'seq_register':
           0x02,'control_register': 0x03,'pin_configuration':0x04, 'adc_reset': 0x0F, 'power_down':0x0B}
19
20     def __init__(self, channels, device_number=0):
21
22
23         self.bus = smbus.SMBus(device_number)
24         time.sleep(ADC._WAIT_TIME)
25         self.channels=ADC.channel_to_byte(channels) #channels va a ser una lista con los numeros de los
               canales ej [1,7]
26         self.num_channels=channels
```

Código C.2: En esta figura se muestra la librería implementada para la configuración y gestión del ADC AD5593. Segunda parte.

```
27
28
29      def configure_channels(self):
30          '''comunica al adc que : quiere escribir, escribe el PB de configuracion de pines de canales adc,
                escribe en MSB y LSB los canales correspondientes'''
31
32          self.bus.write_i2c_block_data(ADC.I2C_ADDR,ADC.PB['pin_configuration'],[0x00,self.channels])
33
34
35      def configure_vref(self):
36
37          self.bus.write_i2c_block_data(ADC.I2C_ADDR,ADC.PB['control_register'],[0x00,ADC._BIT_2_VREF])
38
39
40      def config_secuencia(self):
41          self.bus.write_i2c_block_data(ADC.I2C_ADDR,ADC.PB['seq_register'],[0x00,self.channels])
42
43      def config_repeticion(self):
44          '''comunica al adc que los canales sean de repeticion'''
45
46          self.bus.write_i2c_block_data(ADC.I2C_ADDR,ADC.PB['seq_register'],[ADC._BIT_REP,self.channels])
47
48      def config_adc_buffer(self):
49          '''comunica al adc que los canales sean de repeticion'''
50
51          self.bus.write_i2c_block_data(ADC.I2C_ADDR,ADC.PB['control_register'],[ADC._BIT_BUFFER,0x00])
52
53      def config_power_down(self):
54          '''comunica al adc que los canales sean de repeticion'''
55
56          self.bus.write_i2c_block_data(ADC.I2C_ADDR,ADC.PB['power_down'],[ADC._BIT_PD,self.channels])
57
58      def leer_data(self):
59          '''comunica al adc que quiere leer datos '''
60
61          data=self.bus.read_i2c_block_data(ADC.I2C_ADDR,ADC.PB['adc_readback'])
62          i =0
63          datos = {ch: [0x00,0x00] for ch in self.num_channels}
64          for ch in self.num_channels:
65              datos[ch] = [data[i],data[i+1]]
66              i = i+2
67          datos = self.depurarDatos(datos)
68
69          return ADC.voltaje(datos)
70
71      def depurarDatos(self,datos):
72          '''coprueba que D15 sea 0 (indica que el resultado es ADC data), coprueba que los canales de los
                que vienen los datos son los correctos y devuelve los datos depurados'''
73
74          for ch in datos.keys():
75              if ADC.aplicarMascaraADC15(datos[ch][0])==False:
76                  #lanzar excepcion NO D15=0
```

Código C.3: En esta figura se muestra la librería implementada para la configuración y gestión del ADC AD5593. Tercera parte.

```
77                  print(datos[ch][0])
78                  print("no_cumple_D15")
79                  break
80              if ADC.aplicarMascaraADC1412(datos[ch][0],ch)==False:
81                  #lanzar excepcion NO COINCIDEN LOS CANALES
82                  print("no_concuerda_con_los_canales")
83                  break
84
85              msbDep = ADC.aplicarMascaraADCMSB(datos[ch][0])
86              datos[ch][0]=msbDep
87          return datos
88
89      def resetADC(self):
90          '''establace todos los valores del adc a los valores por defecto'''
91
92          self.bus.write_i2c_block_data(ADC.I2C_ADDR,ADC.PB['adc_reset'],[0x0D,0xAC])
93
94      def setupADC(self):
95          self.configure_channels()
96          self.config_adc_buffer()
97          self.config_power_down()
98          self.config_secuencia()
99
100     @staticmethod
101     def voltaje(data):
102         medidas = {ch: 0 for ch in data.keys()}
103         for ch in medidas.keys():
104             aux = data[ch][0]*2**8
105             total = aux + data[ch][1]
106             total= total*ADC._CONVERSION
107             medidas[ch]=total
108         return medidas
109
110     @staticmethod
111     def channel_to_byte(channels):
112         num=0
113         for i in range(len(channels)):
114             num=num + 2**channels[i]
115         return num
```

Código C.4: En esta figura se muestra la librería implementada para la configuración y gestión del ADC AD5593. Cuarta parte.

```
116
117     @staticmethod
118     def aplicarMascaraADC15(byte):
119
120         if byte<ADC._MASK_D15:
121             return True
122         return False
123
124     @staticmethod
125     def aplicarMascaraADC1412(byte,ch):
126
127         b = byte >> 4
128         if b == ch:
129             return True
130         return False
131
132     @staticmethod
133     def aplicarMascaraADCMSB(byte):
134         ch = byte >> 4
135         ch = ch*(2**4)
136         return byte-ch
```

C.2. Librería para la configuración y gestión del DAC AD9833

Código C.5: En esta figura se muestra la librería implementada para la configuración y gestión del DAC AD9833. Primera parte.

```
1  class AD9833:
2
3      def __init__(self, data, clk, fsync, fmclk):
4          self.dataPin = gpiozero.OutputDevice(pin = data)
5          self.clkPin = gpiozero.OutputDevice(pin = clk)
6          self.fsyncPin = gpiozero.OutputDevice(pin = fsync)
7
8          self.fsyncPin.on()
9          self.clkPin.on()
10         self.dataPin.off()
11
12         self.clk_freq = fmclk #2.4MHz
13         self.setupCLock(self.clk_freq)
14
15     def set_freq(self, f):
16         flag_b28 = 1 << 13 #D13 de controil register
17         flag_freq = 1 << 14 #registro de frecuencia 0, D15=0 D14=1 => FREQ0
18
19         scale = 1 << 28
20         n_reg = int(f *scale / self.clk_freq)
21
22         n_low = n_reg & 0x3fff
23         n_hi = (n_reg >> 14) & 0x3fff
24
25         self.send16(flag_b28)
26         self.send16(flag_freq | n_low)
27         self.send16(flag_freq | n_hi)
28
29     def send16(self, n):
30         self.fsyncPin.off()
31
32         mask = 1 << 15
```

Código C.6: En esta figura se muestra la librería implementada para la configuración y gestión del DAC AD9833. Segunda parte.

```
33          for i in range(0, 16):
34
35              self.dataPin.value = bool(n & mask)
36              self.clkPin.off()
37              self.clkPin.on()
38
39              mask = mask >> 1
40
41          self.dataPin.off()
42          self.fsyncPin.on()
43
44      def setupCLock(self,freq):
45
46          command_line = 'gpio mode 22 clock'
47          args = shlex.split(command_line)
48          subprocess.call(args)
49
50          command_line = 'gpio clock 22 '+str(freq)
51          args = shlex.split(command_line)
52          subprocess.call(args)
```

C.3. Librerías para la configuración y gestión del SHT21 y MPL3115

Código C.7: En esta figura se muestra la librería implementada para la configuración y gestión del SHT21. Primera parte.

```
1  class SHT21:
2
3      _SOFTRESET = 0xFE
4      _I2C_ADDRESS = 0x40
5      _TRIGGER_TEMPERATURE_NO_HOLD = 0xF3
6      _TRIGGER_HUMIDITY_NO_HOLD = 0xF5
7      _STATUS_BITS_MASK = 0xFFFC
8
9      I2C_SLAVE = 0x0703 #cambia la direccion de memoria
10     I2C_SLAVE_FORCE = 0x0706
11
12
13     _TEMPERATURE_WAIT_TIME = 0.086 # (datasheet: typ=66, max=85)
14     _HUMIDITY_WAIT_TIME = 0.030 # (datasheet: typ=22, max=29)
15
16     def __init__(self, device_number=0):
17
18         self.i2c = open('/dev/i2c-%s' % device_number, 'r+', 0) #self,i2c es un atributo que es un
               descriptor de fichero
19         fcntl.ioctl(self.i2c, self.I2C_SLAVE, 0x40)
20         time.sleep(0.050)
21
22     def read_temperature(self):
23
24         self.i2c.write(chr(self._TRIGGER_TEMPERATURE_NO_HOLD))
25         time.sleep(self._TEMPERATURE_WAIT_TIME)
26         data = self.i2c.read(3)
27         if self._calculate_checksum(data, 2) == ord(data[2]):
28             return self._get_temperature_from_buffer(data)
29
30     def read_humidity(self):
31
32         self.i2c.write(chr(self._TRIGGER_HUMIDITY_NO_HOLD))
33         time.sleep(self._HUMIDITY_WAIT_TIME)
34
35         data = self.i2c.read(3) #tres primeros bytes: datos msb datos lsb y cyhecksum
36
37         if self._calculate_checksum(data, 2) == ord(data[2]):
```

Código C.8: En esta figura se muestra la librería implementada para la configuración y gestión del SHT21. Segunda parte.

```
38              return self._get_humidity_from_buffer(data)
39
40      def close(self):
41
42          self.i2c.close()
43
44      def __enter__(self):
45
46          return self
47
48      def __exit__(self, type, value, traceback):
49
50          self.close()
51
52      @staticmethod
53      def _calculate_checksum(data, number_of_bytes):
54
55          POLYNOMIAL = 0x131 # //P(x)=x^8+x^5+x^4+1 = 100110001
56          crc = 0
57          for byteCtr in range(number_of_bytes):
58              crc ^= (ord(data[byteCtr]))
59              for bit in range(8, 0, -1):
60                  if crc & 0x80:
61                      crc = (crc << 1) ^ POLYNOMIAL
62
63                  else:
64                      crc = (crc << 1)
65
66          return crc
67
68      @staticmethod
69      def _get_temperature_from_buffer(data):
70
71          unadjusted = (ord(data[0]) << 8) + ord(data[1])
72          unadjusted &= SHT21._STATUS_BITS_MASK
73          unadjusted *= 175.72
74          unadjusted /= 1 << 16
75          unadjusted -= 46.85
76          return unadjusted
77
78
79      @staticmethod
80      def _get_humidity_from_buffer(data):
81
82          unadjusted = (ord(data[0]) << 8) + ord(data[1])
83          unadjusted &= SHT21._STATUS_BITS_MASK
84          unadjusted *= 125.0
85          unadjusted /= 1 << 16
86          unadjusted -= 6
87          return unadjusted
```

Código C.9: En esta figura se muestra la librería implementada para la configuración y gestión del MPL3115. Primera parte.

```
1  class MPL3115():
2      ADDRESS = 0x60
3
4      def __init__(self,bus):
5          self.i2c = smbus.SMBus(bus)
6
7          if (self.IIC_Read(0x0c) != 196):
8            print("Sensor not detected!")
9      def IIC_Read(self,reg):
10         return self.i2c.read_byte_data(self.ADDRESS, reg)
11
12     def IIC_Write(self,reg,data):
13         self.i2c.write_byte_data(self.ADDRESS, reg, data)
14
15     def config(self,altitude,barometric_pressure):
16         self.altitude = altitude
17         if altitude == True:
18             self.IIC_Write(0x26, 0xb9)
19         else:
20             self.IIC_Write(0x26, 0x39)
21
22         self.IIC_Write(0x29, 0x80)
23         self.IIC_Write(0x13, 0x07)
24
25         pressure = barometric_pressure/2
26
27         self.IIC_Write(0x14, pressure >> 8)
28         self.IIC_Write(0x15, pressure & 0xff)
29
30     def read_data(self):
31         clock = time.clock()
32         out = {'time':time.time()}
33         #out={}
34         #self.wait_new()
35
36         if self.altitude == True:
37             m_altitude = self.IIC_Read(0x01)
38             c_altitude = self.IIC_Read(0x02)
```

Código C.10: En esta figura se muestra la librería implementada para la configuración y gestión del MPL3115. Segunda parte.

```
            l_altitude = float(self.IIC_Read(0x03)>>4)/16.0
            altitude = (m_altitude << 8) | c_altitude
            if altitude > 32768:
                altitude = 0-(65536-altitude)
            out['alt'] = float(altitude) + l_altitude
        else:
            pressure_l = self.IIC_Read(0x03)
            out['pressure'] = self.IIC_Read(0x01) << 10 | self.IIC_Read(0x02) << 2 | pressure_l >> 6

        m_temp = self.IIC_Read(0x04) # temp, degrees
        l_temp = float(self.IIC_Read(0x05)>>4)/16.0 #temp, fraction of a degree
        out['temp'] = m_temp + l_temp
        return out

    def wait_new(self): # todo: add interrupt pin support when we put it in hw.
        while self.IIC_Read(0x12) == False:
            time.sleep(0.5)
```