



Desarrollo de un prototipo *Open Source* para evaluar la función pulmonar en animales pequeños de experimentación

Estudiante: Ezequiel Ignacio Canay

Supervisor: Dr. Ignacio Fenoy

Co-Supervisor: Dr. Ing. Leonardo Casal

Lugar de trabajo: ITECA, ECyT-UNSAM, CONICET

Fecha: Mayo 2023



Parte de los desarrollos presentados en el presente trabajo final integrador han sido publicados en los siguientes congresos y libros:

Congresos:

- **"Desarrollo de un dispositivo para evaluar función pulmonar en ratones: diseño y primeras mediciones"**

Canay, Ezequiel Ignacio; Casal, Leonardo; Fenoy, Ignacio Martín

Congreso Argentino de Sistemas Embebidos – CASE 2021

- **"Development of an open-source device to assess lung function in small laboratory animals"**

Canay, Ezequiel Ignacio; Casal, Leonardo; Fenoy, Ignacio Martín

XXIII Congreso Argentino de Bioingenieria - SABI 2022

Capítulo de libro:

- **"Development of an open-source device to assess lung function in small animals"**

Canay, Ezequiel Ignacio; Casal, Leonardo; Fenoy, Ignacio Martín

IFMBE Proceedings Book Series - International Federation for Medical and Biological Engineering.

Aceptado para su publicación 2023.

Resumen

En este proyecto se desarrolló un prototipo de sistema *Open Source* capaz de medir la función pulmonar de modelos murinos, lo que es esencial en los ensayos pre-clínicos de nuevas terapias para enfermedades pulmonares. Se empleó una bomba de infusión de jeringa impresa en 3D y sensores comerciales de presión y flujo para medir la presión, el flujo y calcular el volumen pulmonar. Tras la calibración del prototipo, se midieron los bucles de presión-volumen pulmonar en ratones sanos, calculando la *Compliance* y Capacidad Inspiratoria. Además, el sistema se validó utilizando dos modelos de asma, demostrando su capacidad para diferenciar entre diferentes condiciones pulmonares. Finalmente se lo validó evaluando la función pulmonar de una rata. El diseño de *Open Source* del prototipo permite su replicación y adaptación a diferentes entornos de investigación, lo que lo convierte en una herramienta valiosa para la comunidad científica.

Palabras clave— Función Pulmonar, *Compliance* Pulmonar, Capacidad Inspiratoria, *Open Source*, Enfermedades Pulmonares, Asma, Modelos Animales, Ratón, Rata.

Development of an open source prototype to assess lung function in small laboratory animals

Abstract

In this project a prototype Open Source system capable of measuring lung function in murine models was developed, which is essential for preclinical trials of new therapies for lung diseases. A 3D-printed syringe infusion pump and commercial pressure and flow sensors were used to measure lung pressure, flow, and calculated volume. After calibration of the prototype, the pressure versus lung volume loops were measured in healthy mice, calculating Compliance and Inspiratory Capacity. Moreover, the system was validated using two models of asthma, demonstrating its ability to differentiate between different lung conditions. Final validation included the evaluation of a rat lung function. The Open Source design of the prototype allows for replication and adaptation to different research environments, making it a valuable tool for the scientific community.

Keywords— Pulmonary Function, Pulmonary Compliance, Inspiratory Capacity, Open Source, Lung Diseases, Asthma, Animal Models, Mice, Rat.

Glosario

ANMAT	Administración Nacional de Medicamentos, Alimentos y Tecnología Médica
CESyMA	Centro de Estudios en Salud y Medio Ambiente
CICUAE	Comité Institucional para el Cuidado y Uso de Animales de Experimentación
CSV	<i>Comma-Separated Values</i>
ECyT	Escuela de Ciencia y Tecnología
EPOC	Enfermedad Pulmonar Obstructiva Crónica
FPI	Fibrosis Pulmonar Idiopática
I ² C	<i>Inter-Integrated Circuit</i>
IC	<i>Inspiratory Capacity</i>
IP	Intraperitoneal
NIH	<i>National Institute of Health</i>
OVA	Ovoalbúmina
P _{el}	Presión elástica
P _{in}	Presión inertiva
P _{res}	Presión resistiva
PAS	<i>Periodic Acid Schiff</i>
PBS	<i>Phosphate Buffered Saline</i>
slm	<i>Standard Litre per Minute</i>
UART	<i>Universal Asynchronous Receiver-Transmitter</i>
UNSAM	Universidad Nacional de San Martín
VFSS	<i>Full Scale Span Voltage</i>

Índice general

Resumen	III
Abstract	v
Glosario	vii
Índice de figuras	xI
Índice de tablas	xIII
1. Introducción	1
1.1. Enfermedades respiratorias	1
1.2. Fisiología Respiratoria	3
1.2.1. Modelo de la mecánica pulmonar	3
1.2.2. Volumen pulmonar	5
1.2.3. Distensibilidad pulmonar	6
1.3. Investigación pre-clínica	8
1.3.1. Modelos animales	9
1.3.2. Anatomía y fisiología pulmonar comparada	9
1.3.3. Desafío de la evaluación de la función pulmonar en animales pequeños de experimentación	11
1.4. Sistemas de medición de función pulmonar	12
1.4.1. Pletismografía sin restricciones	12
1.4.2. Limitaciones de la pletismografía sin restricciones	13
1.4.3. Dispositivos comerciales invasivos	13
1.5. Filosofía <i>open source</i>	14
1.5.1. Principios	14
1.5.2. Ventajas	15
1.5.3. <i>Open Hardware</i>	15
2. Objetivos	17
2.1. Objetivos específicos	17
3. Desarrollo del prototipo	19
3.1. Componentes y funcionamiento	20
3.2. Desarrollo de los programas de adquisición, visualización y procesamiento	26
4. Calibración y validación del prototipo	31
4.1. Calibración de los sensores de presión y flujo	31
4.1.1. Calibración del sensor de presión	31
4.1.2. Verificación de pérdidas de presión	32
4.1.3. Calibración del sensor de flujo	32
4.2. Validación en animales	33

4.2.1.	Condiciones de mantenimiento y cuidado de los animales de experimentación	33
4.2.2.	Procedimiento para medición en ratones	34
4.3.	Validación con modelos de patologías respiratorias	39
4.3.1.	Modelo de asma agudo	39
4.3.2.	Modelo de asma crónico	43
4.4.	Validación con otro modelo animal de experimentación	45
4.4.1.	Procedimiento para medición en rata	45
5.	Discusión	49
6.	Conclusiones	57
Bibliografía		59
Anexos		65
Anexo programas		66
Programa de Experimentación		66
Programa de Procesamiento		74
Firmware del Arduino Uno		84
Anexo hoja de datos		91
MPX5-01ODP (NXP Semiconductors, EE. UU.)		91
FM3400-33-D (Sensirion, Suiza)		100
A4988 (Pololu, EE.UU.)		109

Índice de figuras

1.1.	Mortalidad por enfermedades respiratorias crónicas por país.	2
1.2.	Relaciones presión-volumen (P-V) del sistema respiratorio personas sanas y con EPOC.	6
1.3.	Curvas de presión-volumen espiratorias estáticas de un sujeto asmático antes y después de la aplicación de Salbutamol en aerosol como broncodilatador.	7
1.4.	Representación de una curva de presión-volumen de rango completo en un ratón.	8
1.5.	Dispositivos comerciales invasivos	14
3.1.	Diagrama de bloques del diseño del dispositivo.	20
3.2.	Originales y modificaciones de las piezas 3D de la bomba de infusión.	22
3.3.	Imagen de la bomba de jeringa.	22
3.4.	Diagrama de flujo, control de máquina de estados.	24
3.5.	Imagen de la placa complementaria.	24
3.6.	Esquemático del circuito electrónico.	25
3.7.	Diseño del circuito impreso de la placa <i>Shield</i> para Arduino Uno.	25
3.8.	Interfaz gráfica del Programa de Experimentación.	27
3.9.	Interfaz gráfica del Programa de Procesamiento.	28
4.1.	Mediciones de calibración de los sensores de presión.	32
4.2.	Mediciones para la calibración del sensor de flujo.	33
4.3.	Proceso de eutanasia, pesaje y traqueostomía.	34
4.4.	Posicionamiento del catéter en el pulmón,	35
4.5.	Mediciones representativas de bucles presión-volumen correspondientes a un ratón normal.	36
4.6.	Mediciones representativas de bucles presión-volumen, <i>compliance</i> e IC correspondientes a tres ratones normales.	38
4.7.	<i>Compliance</i> e IC correspondientes a ratones normales de diferentes semanas de edad.	39
4.8.	Modelo de asma agudo.	40
4.9.	Histología de pulmón (tinción con PAS-Hematoxilina).	41
4.10.	Mediciones representativas de bucles presión-volumen correspondientes a un ratón normal y a un ratón del modelo de asma agudo.	42
4.11.	<i>Compliance</i> e IC correspondientes a ratones normales y a ratones del modelo de asma agudo.	43
4.12.	Modelo de asma crónico.	43
4.13.	Mediciones representativas de bucles presión-volumen correspondientes a un ratón normal y un ratón del modelo de asma crónico.	44
4.14.	<i>Compliance</i> e IC correspondientes a ratones normales y a ratones del modelo de asma crónico.	45
4.15.	Mediciones representativas de bucle presión-volumen de una rata.	46

Índice de tablas

3.1. Tabla de costos.	26
-------------------------------	----

Capítulo 1

Introducción

1.1. Enfermedades respiratorias

Las afecciones respiratorias crónicas comprenden un grupo de enfermedades que comprometen a una o más partes del aparato respiratorio, pudiendo afectar los bronquios, los pulmones o la caja torácica [1]. Las enfermedades pulmonares se encuentran entre la tercera causa de mortalidad a nivel mundial después de las enfermedades cardiovasculares y las neoplasias [1]. En 2017 las muertes debidas a enfermedades respiratorias crónicas aumentaron un 18 % respecto de 1990 [1]. La situación de Argentina es particularmente preocupante; de acuerdo con la Organización Panamericana de la Salud la tasa de mortalidad estandarizada por edad es la segunda más alta en las Américas siendo de 56 defunciones por 100.000 habitantes (Figura: 1.1) [2]. Estos datos, sumados a la pandemia de COVID-19, ponen en evidencia la urgente necesidad del estudio de estas patologías respiratorias. Entre ellas se encuentran, la Enfermedad Pulmonar Obstructiva Crónica (EPOC), el asma y la fibrosis pulmonar [1].

El asma es una enfermedad de las vías aéreas inferiores caracterizada por una inflamación crónica, cuyas manifestaciones clínicas son heterogéneas y variables en el tiempo [3]. Los síntomas incluyen sibilancias, dificultad respiratoria, opresión torácica y tos. Si bien el origen de la enfermedad es multifactorial y su etiología no está completamente determinada, se sabe que

involucra la inflamación, la obstrucción intermitente y la hiperreactividad (incremento en la respuesta broncoconstrictora) de las vías respiratorias. La presencia de edema y secreción de mucosidad contribuye tanto con la obstrucción como con el aumento de reactividad. La enfermedad puede tener un curso agudo, subagudo o crónico, dependiendo de la frecuencia, duración e intensidad de sus manifestaciones clínicas [3].

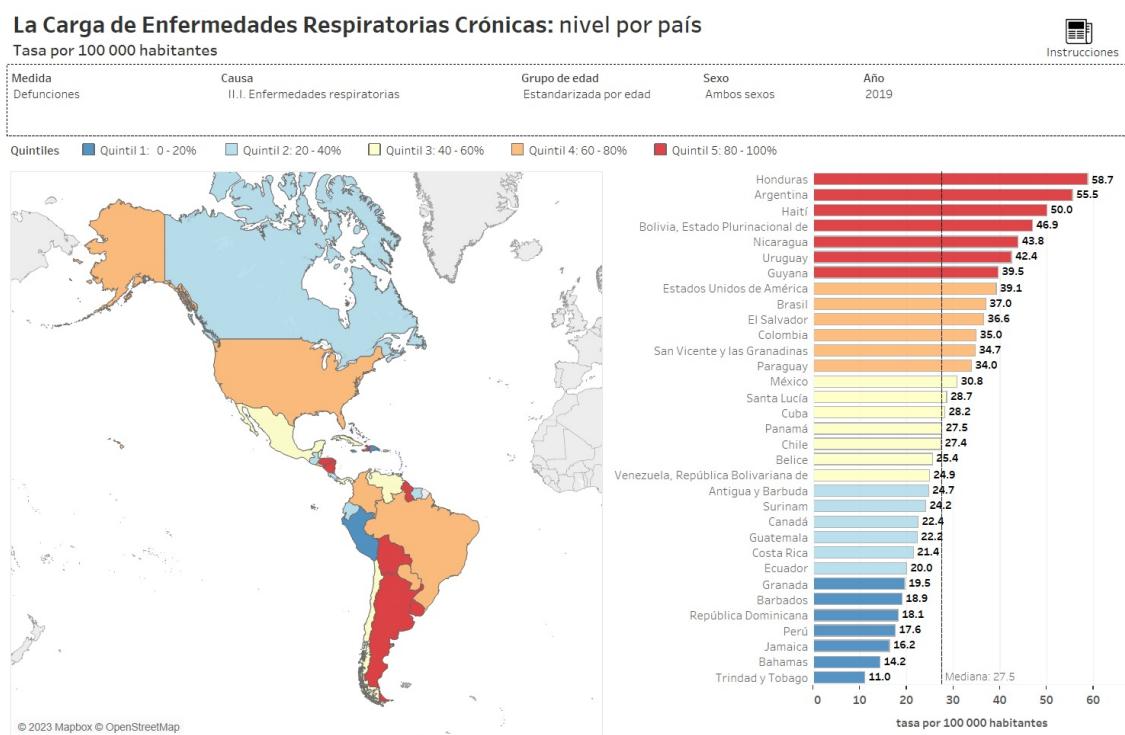


Figura 1.1: Mortalidad por enfermedades respiratorias crónicas por país.
Gráfico generado por Organización Panamericana de la Salud [2].

La Enfermedad Pulmonar Obstructiva Crónica (EPOC) es un trastorno pulmonar que se caracteriza por la existencia de una obstrucción de las vías respiratorias generalmente progresiva e irreversible [4]. Se encuentra una mayor incidencia en personas expuestas al humo del tabaco, leña o carbón. Produce como síntoma principal una disminución de la capacidad respiratoria, que avanza lentamente con el paso de los años y ocasiona un deterioro considerable en la calidad

de vida de las personas afectadas, pudiendo ocasionar una muerte prematura. Entre el 20 % y el 25 % de los fumadores desarrolla la enfermedad, pero se desconocen los factores predisponentes al desarrollo, aunque puede que sea un componente multifactorial que incluya elementos ambientales y factores genéticos [4].

La Fibrosis Pulmonar Idiopática (FPI) es una enfermedad crónica que se caracteriza por una disminución progresiva de la función pulmonar. El término fibrosis pulmonar hace referencia a la cicatrización del tejido pulmonar y es la causa del agravamiento de la disnea (dificultad respiratoria) [5]. La fibrosis idiopática es mortal. Luego de su diagnóstico los pacientes tienen una esperanza de vida que ronda entre los 2,5 y 3,5 años de vida. El término «idiopático» se utiliza debido a que aún se desconoce la causa de la fibrosis pulmonar. Normalmente la FPI afecta a adultos de entre 50 y 70 años, en particular a aquellos con antecedentes de tabaquismo, y en mayor número a los hombres que a las mujeres [5].

1.2. Fisiología Respiratoria

La fisiología respiratoria se ocupa del estudio del funcionamiento del sistema respiratorio y de los procesos que permiten la oxigenación de la sangre. Las pruebas de función pulmonar son herramientas importantes para describir las características fenotípicas de una enfermedad respiratoria [6]. Estas pruebas se basan en un modelado previo de la función pulmonar.

1.2.1. Modelo de la mecánica pulmonar

Para llevar adelante la evaluación del funcionamiento del aparato respiratorio, especialmente la evaluación de la mecánica pulmonar, es necesario emplear un modelado del pulmón [7, 8]. El modelo más simple es un tubo conectado a un fuelle. Este modelo reproduce bien el comportamiento fisiológico para una sola frecuencia respiratoria, pero tiene importantes limitaciones cuando se consideran los cambios en la mecánica pulmonar que ocurren con altera-

ciones en la frecuencia respiratoria. Esto se debe a que las propiedades resistivas y elásticas del pulmón dependen sustancialmente de la frecuencia respiratoria. Por ejemplo, la resistencia del pulmón cae a medida que aumenta la frecuencia respiratoria por encima del rango asociado con la respiración normal [9]. Para modelar este tipo de comportamiento mecánico, es necesario incluir en el modelo de manera conjunta un resorte y un amortiguador capaces de simular el comportamiento viscoelástico del pulmón. Estos modelos básicos permiten desarrollar expresiones matemáticas que se pueden utilizar para evaluar cuantitativamente la mecánica pulmonar. Los parámetros de los modelos, es decir, los valores resistivos y elásticos de sus componentes individuales, constituyen los puntos finales que se utilizan para evaluar experimentalmente la función pulmonar. El modelo viscoelástico describe sustancialmente mejor la mecánica pulmonar de naturaleza dependiente de la frecuencia que el modelo de tubo y fuelle. Sin embargo, este modelo más simple todavía sirve como plataforma conceptual para la mayoría de los estudios de la mecánica pulmonar y la capacidad de respuesta bronquial [10]. El comportamiento mecánico de este modelo se describe mediante su ecuación de movimiento. Esta ecuación de la física clásica establece que la fuerza aplicada al modelo es igual y opuesta a la fuerza que genera el modelo. Si definimos a la presión como la fuerza ejercida por unidad de área, podemos inferir que la presión aplicada (P) es igual y opuesta a la presión sumistrada por los músculos respiratorios. La presión de oposición se compone, en general, de tres componentes: una presión resistiva (P_{res}), una presión elástica (P_{el}) y una presión inertiva (P_{in}).

De este modo,

$$P = P_{res} + P_{el} + P_{in} \text{ (Ecuación 1)}$$

P_{res} se describe por la ley de Ohm:

$$P_{res} = R\varphi \text{ (Ecuación 2)}$$

donde R es la resistencia del pulmón y φ es el flujo de gas.

P_{el} es descrita por la ley de Hooke:

$$P_{el} = EV = \frac{1}{c}V \text{ (Ecuación 3)}$$

donde E es la elastancia pulmonar, igual a la inversa de la distensibilidad o *compliance* (C), y V es el volumen pulmonar relativo a la capacidad residual funcional.

Por otro lado, P_{in} entra en juego solo en frecuencias muy por encima de las de la respiración normal. Del mismo modo, P_{res} se vuelve insignificante cuando la frecuencia es extremadamente baja. Por lo tanto, la ecuación de movimiento relevante para la respiración normal es

$$P = R\varphi + \frac{1}{c}V \text{ (Ecuación 4)}$$

Los parámetros R y E dependen profundamente de la frecuencia respiratoria y del volumen pulmonar.

1.2.2. Volumen pulmonar

Para evaluar la mecánica pulmonar, otro parámetro relevante es el volumen de los pulmones. En este sentido, éste influencia de manera importante la relación entre presión y flujo. Por ejemplo, un aumento en el volumen pulmonar abre las vías respiratorias y hace que disminuya la resistencia de las mismas (anclaje). Esto también dificulta que las vías respiratorias se estrechen cuando el músculo liso de éstas se contrae [7, 11].

La Capacidad Inspiratoria (IC)¹ en humanos se define por el esfuerzo voluntario máximo que una persona puede hacer, pero desafortunadamente esto no se puede replicar en ningún modelo animal. Por lo tanto, el volumen máximo en las curvas presión-volumen experimentales se determina mediante una presión máxima arbitrariamente establecida por el investigador [12].

¹IC por sus siglas en inglés *Inspiratory Capacity*

1.2.3. Distensibilidad pulmonar

Otro parámetro relevante para evaluar la función pulmonar es la distensibilidad pulmonar. Este parámetro refleja la facilidad con la que puede expandirse y contraerse el órgano, y se relaciona con la elasticidad del mismo. En los bucles de distensibilidad (presión-volumen) se puede observar el componente de la pérdida de presión transpulmonar que está fuera de fase con el flujo y en fase con el volumen, así como la presión de retroceso ejercida por el pulmón en condiciones estáticas, son causados por las fuerzas elásticas dentro del pulmón [13]. La pérdida de retroceso elástico dentro del pulmón define el enfisema, mientras que un aumento define los procesos restrictivos [7, 14]. El retroceso elástico del pulmón se evalúa en términos de la curva de presión-volumen cuasiestática, medida inflando y desinflando el pulmón de forma escalonada. La rama inspiratoria de la curva sigue un camino a través de valores de P que son más altos que los de la rama espiratoria, como se ve en la figura 1.2 (A), denominándose histéresis a la diferencia entre las dos ramas. Los cambios en la rama inspiratoria de la curva presión-volumen que causan un aumento en la histéresis se toman para indicar un mayor cierre de las vías respiratorias, como el observado en humanos después de la inhalación de gas frío seco [15].

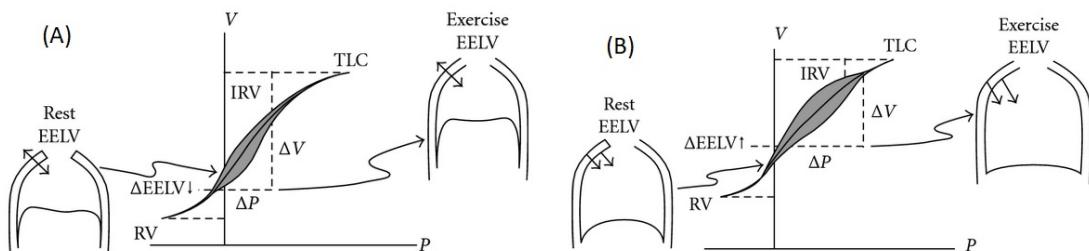


Figura 1.2: Relaciones presión-volumen (P-V) del sistema respiratorio personas sanas y con EPOC. En (A) normal y (B) enfermedad pulmonar obstructiva crónica (EPOC). En los individuos con EPOC hay un reseteo del volumen de relajación del sistema respiratorio a un nivel más alto que en los individuos sanos. La hiperinflación en la EPOC conduce a un aumento del volumen pulmón final espiratorio (EELV), volumen residual (RV) y una reducción correspondiente en el volumen de reserva inspiratoria (IRV), en comparación con la condición normal. En contraste con el pulmón normal, la presión de recuperación combinada de los pulmones y la pared torácica en la Hiperinsuflación es dirigida hacia adentro durante el reposo y durante el ejercicio. Esto resulta en una carga de umbral inspiratorio en los músculos inspiratorios con una consecuente disminución en la zona de aposición (mostrada en la curva P-V (B) durante el reposo y el ejercicio). EELV: volumen pulmón final espiratorio, RV: volumen residual, IRV: volumen de reserva inspiratoria, TLC: capacidad total del pulmón. Extraído de Papandrinopoulou *et al.* [16].

Con respecto a pacientes con asma, Holmes *et al.* y Colebatch *et al.* observaron que el extremo inferior de la curva presión-volumen se alteró sin un cambio acompañante de la capacidad pulmonar total [17, 18]. La pendiente reducida del extremo inferior de la curva presión-volumen se atribuye al cierre de las vías respiratorias con volúmenes pulmonares bajos [19, 18] (Figura: 1.3). Esto se revertiría a medida que el asma mejorara después de la aplicación de un broncodilatador. Además, el cierre de las vías respiratorias estrechadas durante la espiración, con el consiguiente atrapamiento de gas en las unidades pulmonares distales, también produciría un desplazamiento hacia la izquierda del extremo inferior de la curva presión-volumen. Cambios opuestos ocurren después del uso de un broncodilatador [17].

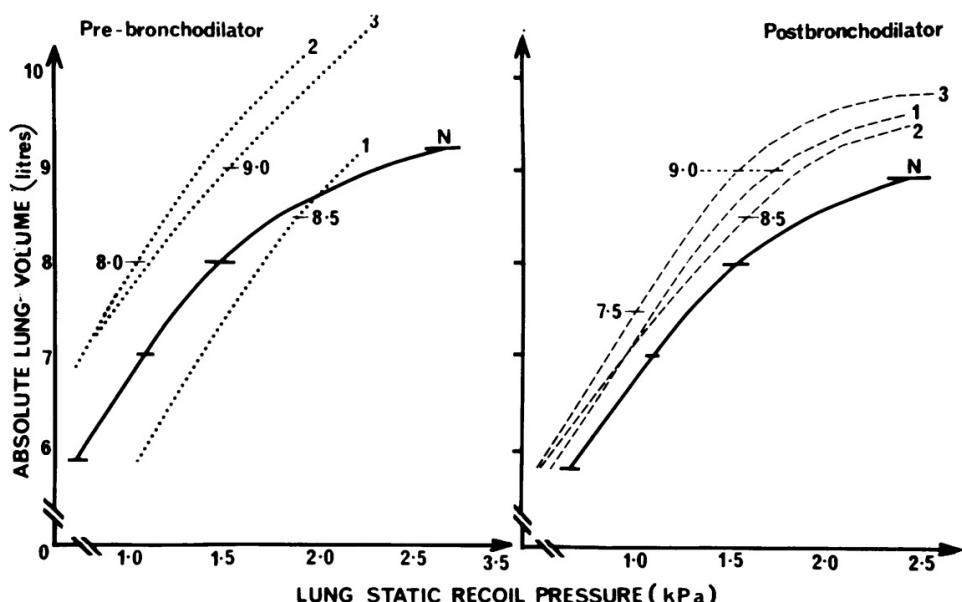


Figura 1.3: Curvas de presión-volumen espiratorias estáticas de un sujeto asmático antes y después de la aplicación de Salbutamol en aerosol como broncodilatador.

Se compara la media de cinco curvas presión-volumen (N) obtenidas después de la respiración normal con las curvas presión-volumen (1-3), cada una de las cuales se ha obtenido después de respirar durante 60 segundos en los volúmenes pulmonares indicados. El número se refiere al orden serial de las pruebas. Extraído de Holmes *et al.* [17].

La *compliance* (C) es un parámetro que se puede obtener de cualquier región linealizada de la curva presión-volumen no lineal. Uno de los problemas críticos al usar cualquier medición de pendiente de la curva presión-volumen es que el valor depende en gran medida tanto del rango

de presión sobre el que se mide como de la historia de volumen anterior (es decir, cómo se generó la sección de la curva), por lo que la consistencia es crucialmente importante si se van a hacer comparaciones entre modelos de control y patológicos [12].

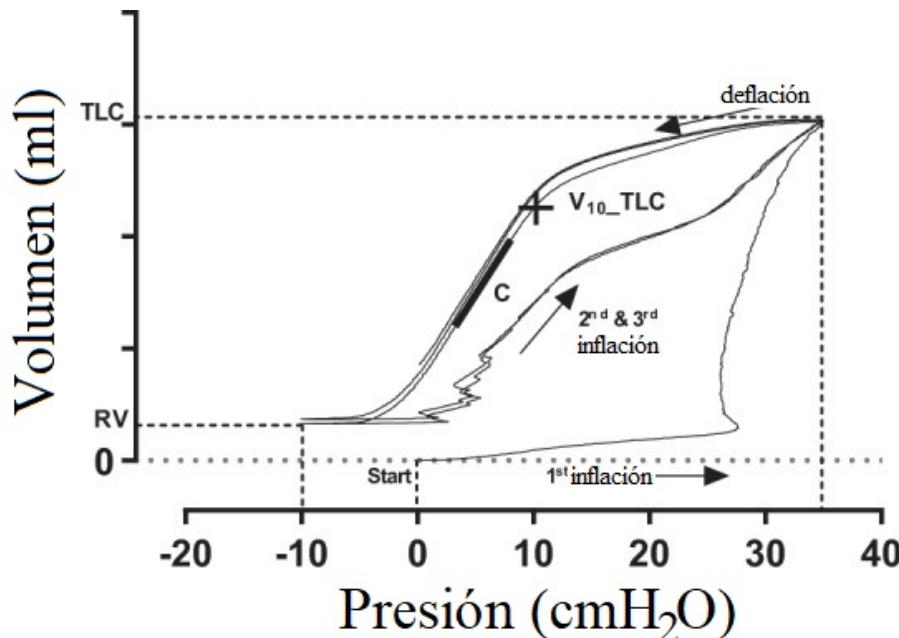


Figura 1.4: Representación de una curva de presión-volumen de rango completo en un ratón.
Se han identificado los parámetros típicamente derivados de ella. TLC, capacidad pulmonar total; RV, volumen residual; C, compliance; V₁₀_TLC, un parámetro que describe la forma definido como el volumen a 10 cmH₂O (V₁₀) normalizado a TLC. Extraído de Robichaud *et al.* [20].

1.3. Investigación pre-clínica

Los estudios pre-clínicos son una parte importante del proceso de desarrollo de nuevas terapias médicas [21]. Antes de que se pueda probar una nueva terapia en humanos, debe pasar por una serie de pruebas en un entorno controlado para asegurar que es seguro y eficaz [21]. Estos estudios se llevan a cabo en modelos animales, como ratones [22], y pueden incluir pruebas de dosis, efectos secundarios y mecanismos de acción [22]. Los resultados de estos estudios pueden ayudar a determinar si una nueva terapia es lo suficientemente segura y eficaz como para ser probada en humanos [22].

1.3.1. Modelos animales

Gran parte de nuestra comprensión actual respecto del funcionamiento del pulmón y de las enfermedades pulmonares proviene de estudios realizados en animales [23, 24, 25, 26]. Entre los modelos pre-clínicos, en la actualidad los ratones se emplean ampliamente en la investigación pulmonar debido a ciertas ventajas que esta especie proporciona, como son: una buena comprensión de su sistema inmunológico, la amplia gama de disponibilidad de reactivos, su ciclo reproductivo corto, un genoma bien caracterizado, el advenimiento de la cepas transgénicas y deficientes, y al bajo costo de producción y mantenimiento. En este sentido, existen modelos experimentales bien definidos en ratones para distintas enfermedades pulmonares como asma, bronquitis crónica, enfisema, EPOC y fibrosis quística entre otros [27, 26]. La medición de la función pulmonar en estos modelos es esencial para establecer su relevancia respecto de la enfermedad pulmonar humana.

1.3.2. Anatomía y fisiología pulmonar comparada

La capacidad pulmonar total (TLC) del ratón es de aproximadamente 1 ml, mientras que en la rata es de 6 ml y en el humano de 6000 ml. El ratón, al igual que el humano, posee 5 lóbulos en el pulmón derecho, pero a diferencia de éste, sólo posee un único lóbulo en el pulmón izquierdo. Por otro lado, la pleura del ratón y de la rata es delgada, pero lo suficientemente fuerte como para inflarse a presiones considerablemente más altas que los 30 cm H₂O [10].

El parénquima del pulmón del ratón ocupa una fracción menor del pulmón total que la de la rata, pero más que la del humano, siendo estos valores de 18 % para el ratón, 24 % para la rata y 12 % para el humano respecto al volumen pulmonar. Los alvéolos del pulmón del ratón son más pequeños que los de la rata y el humano, presentando una intersección lineal media (MLI) de 80 µm en comparación con los 100 µm de la rata y los 210 µm del humano. El espesor de la barrera alvéolo-capilar en el ratón es similar al de la rata, siendo de 0,32 µm y 0,38 µm respectivamente, pero algo menor que el del ser humano que es de 0,62 µm, lo que podría tener implicaciones

importantes tanto para el intercambio de gases como para la mecánica pulmonar. Las vías respiratorias constituyen un gran porcentaje del pulmón en ratones en comparación con las ratas, siendo de 11 y 5,7% respectivamente.

El cartílago está presente en la tráquea del ratón, pero está menos organizado que en otras especies; sólo la parte superior de la tráquea tiene los anillos completos que se ven en otros mamíferos y estos se transforman rápidamente en placas a medida que se avanza distalmente. Los pulmones de ratón tienen menos bronquiolos respiratorios y generaciones de vías respiratorias que van de entre 13 a 17, mientras que los pulmones humanos van de 17 a 21 generaciones. Las vías respiratorias del pulmón de ratón exhiben un patrón de ramificación monopodial en lugar de dicotómico. Otras dos características importantes del pulmón del ratón son la delgadez del epitelio respiratorio y la luz de las vías respiratorias relativamente grande [28, 29]. Se especula que este gran calibre de las vías respiratorias reduce la carga resistente al flujo que, de otro modo, resultaría de la frecuencia respiratoria rápida (250-350 lpm) requerida por el ratón para mantener la temperatura corporal [30]. Una diferencia funcional importante entre ratones y ratas en comparación con los humanos es la escasez, o incluso la ausencia total, de glándulas submucosas y la presencia de un alto número de células Club productoras de mucus [29]. Si bien el significado exacto que tienen todas estas características del pulmón del ratón en la función pulmonar es desconocido, se ha descripto que la resistencia inicial de las vías respiratorias entre ratones que han sido sensibilizados y desafiados con alérgenos difieren imperceptiblemente de la de los animales sanos y que debe tratarse con agonistas colinérgico como la metacolina para acentuar las diferencias entre animales sanos y tratados [31, 32]. Esto último sugiere que los procesos inflamatorios que comprometen la función pulmonar en animales más grandes como el humano, podrían tener poco efecto en ratones debido al tamaño relativamente más grande de sus vías respiratorias y/o a la falta de glándulas mucosas.

Las paredes y otras estructuras torácicas en ratones son extremadamente flexibles, por lo que la mayor parte del retroceso elástico medido en un animal intacto se puede atribuir específicamente al pulmón. Además, el retroceso elástico del pulmón muestra una variabilidad gené-

tica considerable que debe tenerse en cuenta en los diseños de estudio [33]. En este sentido se ha descripto cambios en el retroceso elástico del pulmón en ratones con inflamación alérgica pulmonar [34]. Estos cambios en las características de la curva presión-volumen pueden ser indicadores sensibles de disfunción pulmonar y contribuir a la génesis de la hiperreactividad. En este sentido, en el ratón las ramas de deflación en la curva presión-volumen de la mayoría de las cepas son bastante lineales entre 3 y 8 cm H₂O, y por esta razón es fácil definir una *compliance* reproducible en ese rango.

1.3.3. Desafío de la evaluación de la función pulmonar en animales pequeños de experimentación

Realizar mediciones en murinos presenta desafíos técnicos particulares debido al pequeño tamaño de los mismos, particularmente en lo que respecta a la medición de los flujos respiratorios [10]. El desarrollo de sistemas para evaluar la función pulmonar que se adapten a las necesidades individuales de las investigaciones permitirá reducir y refinar el uso de animales de experimentación que se emplean en el estudio de estas patologías.

La medición de la capacidad residual funcional por dilución de gas es igualmente difícil, nuevamente debido al pequeño tamaño del pulmón del ratón, y solo hay algunos informes en la literatura sobre el uso de esta técnica [35]. Otros estudios del volumen pulmonar del ratón han utilizado un enfoque de flotabilidad [14], de desgasificación [33, 36] e incluso se ha informado un método de tomografía computarizada [37]. Sin embargo, ninguno de estos es particularmente práctico para la mayoría de los diseños experimentales.

El pulmón del ratón exhibe un comportamiento elástico lineal [15, 34]. El flujo de aire es lineal (es decir, regímenes de flujo laminar), ya que es muy poco probable que se produzca un flujo turbulento en los pulmones de los ratones debido a los diámetros pequeños de las vías respiratorias, a diferencia de los humanos, donde el flujo turbulento es una ocurrencia común [34], lo que implica una limitación de esta especie en la exploración de condiciones complejas

de flujo de aire.

1.4. Sistemas de medición de función pulmonar

Existen diferentes sistemas para evaluar la función pulmonar en animales pequeños de experimentación, tanto invasivos como no invasivos [10, 12, 38, 39]. Entre los métodos no invasivos se destacan por su amplio uso aquellos que emplean pletismógrafos de cuerpo entero. En los abordajes invasivos se realizan maniobras de ventilación forzada en animales anestesiados y traqueostomizados [40].

1.4.1. Pletismografía sin restricciones

Este enfoque para evaluar la función pulmonar implica colocar al sujeto en un pequeño compartimiento cerrado y medir los cambios de presión dentro de la caja que ocurren cuando el animal respira [10, 41, 42]. El animal está consciente y sin restricciones. Esta técnica actualmente goza de gran popularidad [42] porque es simple y el ratón permanece ileso después del experimento. El parámetro que se obtiene con esta técnica es conocido como Penh, que significa “pausa mejorada”. Es importante tener en cuenta que no existe un vínculo entre Penh y otras variables que se derivan de principios mecánicos: Penh es simplemente un derivado empírico de las variaciones respiratorias en la presión de la caja [10]. Mientras que un artículo inicial mostró una correlación razonables entre Penh y medidas invasivas de la mecánica pulmonar [42], publicaciones recientes cuestionan seriamente la validez del uso de Penh para medir la función pulmonar [10, 41, 43]. Los cambios de presión que ocurren dentro de la caja cuando el ratón respira se derivan primero de la compresión y descompresión del gas dentro del tórax, un evento relacionado con el estado de la mecánica pulmonar, y en segundo lugar de la humidificación y el calentamiento del gas inspirado, un evento no relacionado con la mecánica pulmonar. Durante la broncoconstricción, ambos componentes aumentan [41].

1.4.2. Limitaciones de la pletismografía sin restricciones

Estudios recientes muestran que los cambios en Penh se apartan de los cambios mecánicos durante un estado de aumento de la temperatura de la caja [41, 43], de manera exactamente opuesta durante la exposición a condiciones hiperóxicas [43, 44] como a la exposición repetida a alérgenos aerosolizados [45]. Estos hallazgos muestran que Penh no es una medida válida de la función pulmonar del ratón, excepto como medida de los patrones de respiración, y se sabe desde hace mucho tiempo que los patrones de respiración generalmente tienen poca relación con la mecánica pulmonar. Finalmente, una respuesta en Penh también puede deberse a cambios en la resistencia de la cavidad nasal, ya que las vías respiratorias superiores contribuyen de manera muy significativa (50 %) a la resistencia pulmonar total y es probable que su contribución cambie según la situación experimental.

1.4.3. Dispositivos comerciales invasivos

La oferta mundial de aparatos comerciales para medir función pulmonar en roedores se encuentra limitado a dos empresas, la estadounidense DSI que comercializa la marca Buxco® y la canadiense SCIREQ que comercializa la línea FlexiVent® (Figura: 1.5). En la actualidad existe una ausencia de dispositivos de código abierto, simples y de bajo costo que evalúen la fisiología pulmonar.



Figura 1.5: Dispositivos comerciales invasivos
Equipo DSI de la marca Buxco® (izq). FlexiVent® de la marca SCIREQ (der).

1.5. Filosofía *open source*

La filosofía del código abierto u *open source* se basa en el principio de que el conocimiento y el *software* deben ser compartidos de manera abierta y transparente [46]. Esto significa que cualquier persona debe tener acceso al código fuente del programa y ser libre de utilizar, modificar y distribuir ese *software* siempre y cuando respete las condiciones de la licencia *open source* aplicable [47].

1.5.1. Principios

Los principios fundamentales de la filosofía *open source* incluyen la libertad de uso, la libertad de modificación, la libertad de distribución y la libertad de acceso al código fuente. La libertad de uso hace referencia a que cualquier persona puede utilizar el *software* de cualquier manera que desee. La libertad de modificación implica que cualquier persona puede modificar el código fuente del programa para adaptarlo a sus necesidades o para solucionar errores. La libertad de distribución significa que cualquier persona puede distribuir copias del *software*, tanto

en forma gratuita como a cambio de un pago. Finalmente, la libertad de acceso al código fuente significa que cualquier persona puede acceder al código fuente y utilizarlo como base para crear *software* derivado [48].

1.5.2. Ventajas

La filosofía *open source* ha demostrado ser esencial en el avance de la ciencia y la tecnología [49]. Esto se debe a que promueve la colaboración y el compartir conocimiento de manera abierta y transparente [46], lo que permite a los científicos y tecnólogos trabajar juntos de manera más efectiva y rápida [50]. Además, el hecho de que el código y la información estén disponibles para cualquiera permite que más personas puedan contribuir y aportar ideas, lo que enriquece el proceso de investigación y desarrollo [51]. Esto facilita la colaboración y el compartir conocimiento de manera efectiva y transparente

1.5.3. Open Hardware

El llamado *Open Source Hardware* forma parte de esta corriente de pensamiento, siendo un *hardware* cuyo diseño se pone a disposición del público para que cualquier persona pueda estudiar, modificar, distribuir, fabricar y vender el diseño o el *hardware* asociado a ese diseño. Para que un producto se defina como *open source* no se debe tener en cuenta únicamente la licencia, sino también la correcta documentación [52].

De acuerdo a la Open Source Hardware Association (OSHWA), para que un producto puede ser considerado *Open Source Hardware* es necesario que su documentación cumpla con ciertos requisitos [53]. En primer lugar, la documentación debe estar disponible para cualquier persona de manera gratuita y sin restricciones. Esto incluye los planos y especificaciones del hardware, así como también cualquier otro tipo de información relevante, como instrucciones de uso o guías de montaje. En segundo lugar, la documentación debe estar disponible en un formato que pueda ser leído y modificado por cualquier persona. Esto puede incluir archivos de diseño

electrónico (como archivos CAD) o archivos de texto plano (como archivos de código o documentos de texto). Finalmente, la documentación debe estar licenciada bajo una licencia *open source* apropiada. Esto significa que cualquier persona puede utilizar, modificar y distribuir la documentación siempre y cuando respete las condiciones de la licencia [53].

Capítulo 2

Objetivos

Por todo lo expuesto, el presente trabajo final integrador tiene como objetivo desarrollar un prototipo *open source* que mida la presión pulmonar y el flujo respiratorio de manera directa y que permita realizar ventilación automática en animales pequeños de experimentación. Estos parámetros medidos se emplearán para graficar la curva presión volumen y evaluar la función pulmonar mediante la medición de *compliance* y capacidad Inspiratoria en animales pequeños de experimentación.

2.1. Objetivos específicos

1. Desarrollar un prototipo *open source* que permita obtener las curvas de presión-volumen de ratones de laboratorio.
2. Desarrollar un entorno gráfico que permita visualizar en tiempo real las curvas obtenidas.
3. Desarrollar un algoritmo que permita obtener los parámetros de *compliance* y Capacidad Inspiratoria partir de los datos obtenidos.
4. Validar el prototipo con distintos modelos de patologías respiratorias.

Capítulo 3

Desarrollo del prototipo

Con el objetivo de desarrollar un dispositivo que permita evaluar la función pulmonar en animales pequeños de experimentación, se diseño un prototipo que debía ser capaz de medir la presión pulmonar y el flujo respiratorio. Para este diseño decidimos emplear un enfoque *open source* que permite a la comunidad científica acceder a una herramienta útil y económica para llevar adelante estudios pre-clínicos de afecciones pulmonares.

Para alcanzar la versión final del prototipo presentado en el presente informe se trabajó con una metodología de desarrollo ágil e iterativa que permitió realizar mejoras continuas en cada versión anterior del prototipo. Se trabajó en ciclos cortos de desarrollo que incluyeron la definición de requisitos y especificaciones, el diseño y el desarrollo de un prototipo, la experimentación con animales, la evaluación de los resultados y una retroalimentación. Durante cada iteración antes de probar el prototipo en un animal se realizaron verificaciones de funcionamiento utilizando un globo pequeño para agua, a fin de simular un pulmón. En cada ciclo, se recopiló información valiosa de la evaluación y la retroalimentación que permitió generar una nueva versión mejorada del prototipo luego de cada iteración. Como resultado de cinco ciclos de iteración se obtuvo el prototipo actual que se detalla a continuación.

3.1. Componentes y funcionamiento

Se desarrollo un prototipo que permite la medición de la presión pulmonar y el flujo respiratorio mediante sensores comerciales. El volumen respiratorio se calcula mediante la integración del flujo respiratorio.

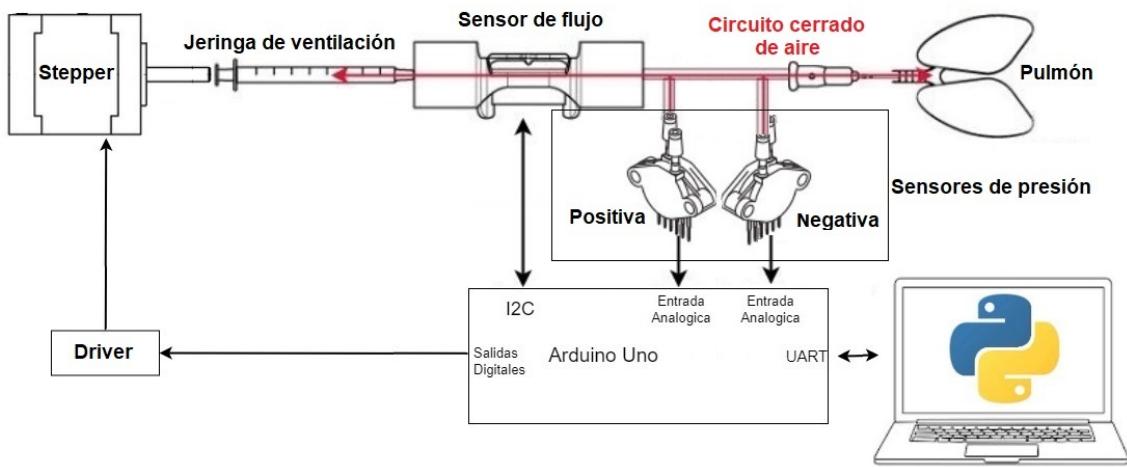


Figura 3.1: Diagrama de bloques del diseño del dispositivo.

Como se muestra en el diagrama de bloques (Figura: 3.1), el pulmón del ratón se conecta a un circuito cerrado de aire mediante una traqueostomía. El circuito cerrado de aire incluye dos sensores de presión en paralelo, un sensor de flujo en serie y una jeringa de 20 ml como punto final. Un motor paso a paso (*Stepper*) controlado por un *driver* mueve el émbolo de la jeringa. El Arduino Uno lee las mediciones de los sensores y se conecta a una PC.

Con el objetivo de medir la presión, se utilizaron dos sensores de presión analógicos MPX5-010DP (NXP Semiconductors, EE. UU.) conectados respectivamente al puerto de presión positiva y al puerto de vacío. De esta forma, un sensor mide la presión positiva y el otro mide la presión negativa, permitiendo obtener la medición completa del bucle respiratorio. Estos sensores ofrecen una precisión de medición del 5 % del VFSS¹ y tienen un rango de medición de 0 a 10 kPa con respecto a la presión ambiente (hoja de datos en el Anexo MPX5-010DP).

¹Full Scale Span Voltage se define como la diferencia algebraica entre el voltaje de salida a la presión nominal máxima y el voltaje de salida a la presión nominal mínima.

Con la finalidad de medir el flujo respiratorio, se utilizó el sensor SFM3400-33-D (Sensirion, Suiza). El sensor permite la medición de un caudal volumétrico con una precisión del 0,5%, una frecuencia de muestreo de 1 kHz y un rango de $\pm 33 \text{ slm}^2$. El sensor utiliza una interfaz I²C para transferir información y proporciona un valor digital con una resolución de 14 bits (hoja de datos en el Anexo FM3400-33-D).

Para llevar a cabo la ventilación mecánica, se decidió adaptar una bomba de infusión a jeringa *open source* de impresión 3D. La misma fue descripta previamente por Bas Wijnen *et al.* [54]. La bomba de infusión se controla mediante un motor paso a paso NEMA 17 y un *driver* para motores paso a paso A4988 (Pololu, EE.UU.). El motor utilizado posee un torque de 2,9 kg/cm. Por otro lado el *driver* empleado permite una resolución de hasta la dieciseisava parte de un paso (hoja de datos en el Anexo A4988).

A fin de alcanzar los requerimientos necesarios para la ventilación, se realizaron diversas modificaciones a la bomba de infusión. En primer lugar, se decidió reemplazar el tornillo rosulado de 5 mm por un tornillo de 8 mm con rosca tipo ACME. En consecuencia a este cambio, se empleó una tuerca de 4 hilos de avance. Esta modificación se llevó a cabo con el objetivo de reducir el coeficiente de fricción, tal como se había reportado previamente por Shriraoy col. [55]. Además, se rediseñó la abrazadera de la jeringa para facilitar el empleo de una nueva jeringa en cada procedimiento (Figura: 3.2 A) . Finalmente, se realizaron las modificaciones necesarias en los diseños de las piezas 3D para que puedan admitir las modificaciones mencionadas (Figura: 3.2). La versión modificada fue impresa, ensamblada y se verificó su correcto funcionamiento.

²Standard Litre per Minute

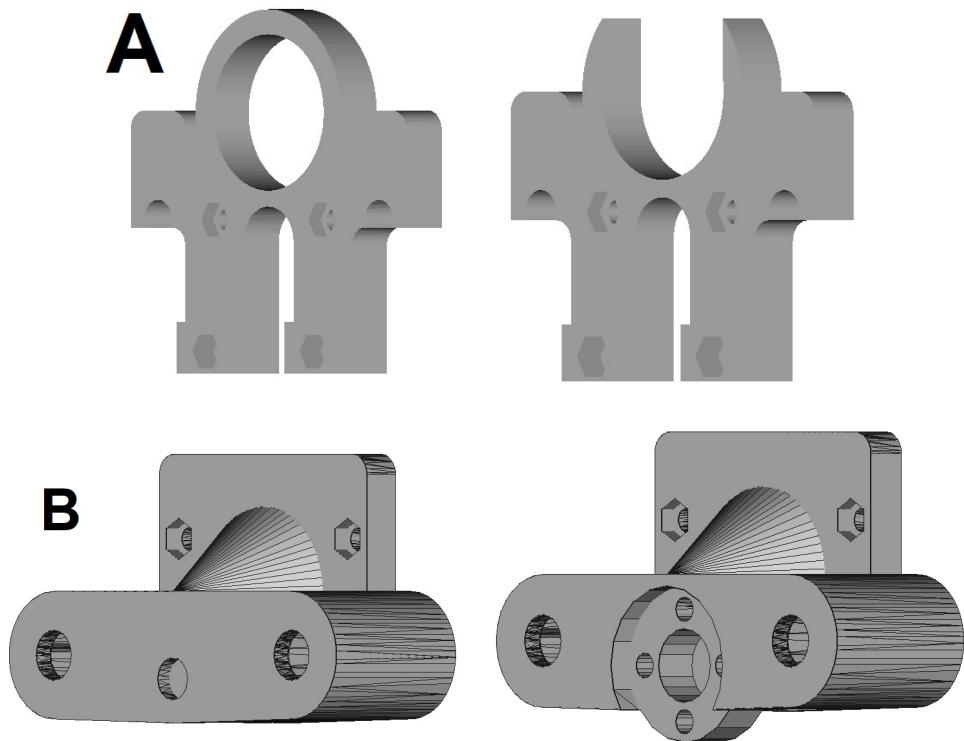


Figura 3.2: Originales y modificaciones de las piezas 3D de la bomba de infusión.

(A) Abrazadera de la jeringa original (izq) y abrazadera de la jeringa modificado (der). (B) Carro que empuja el émbolo original (izq) y carro que empuja el émbolo modificado para admitir tuerca de 4 hilos (der).

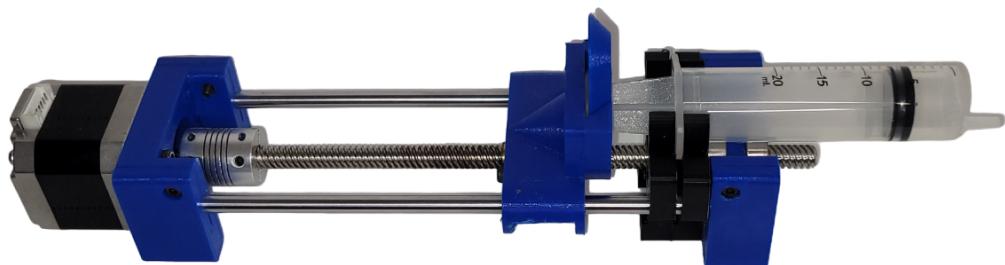


Figura 3.3: Imagen de la bomba de jeringa.

Para controlar la bomba se programó la placa de desarrollo Arduino Uno utilizando un modelo de control por máquina de estados en C++ (Figura: 3.4). El microcontrolador queda a la espera de recibir una comunicación con instrucciones de una PC a través de la interfaz UART³. Al recibirla, carga los parámetros de velocidad de ventilación, de volumen máximo y de presión máxima, e inicia el ciclo de ventilación. A través del *driver* A4988, la placa Arduino controla el motor paso a paso para generar un flujo de inspiración positivo hasta que se alcance alguno de los dos parámetros máximos deseados, ya sea el de volumen o el de presión. A continuación, se genera un flujo de espiración negativo hasta alcanzar el mismo volumen inspirado o la presión mínima de emergencia. Esto último actúa como una protección de emergencia para los pulmones y detiene el proceso en caso de que la presión disminuya por debajo del valor mínimo permitido. Al finalizar el ciclo de ventilación, la placa Arduino revisa si se recibió una nueva comunicación con instrucciones y comienza un nuevo ciclo. Durante todo el proceso, la placa Arduino lee las mediciones de los tres sensores y las transmite a través de la interfaz UART a la PC. En el Anexo Firmware del Arduino Uno se encuentra el código del *firmware*.

Para la conexión de todas las partes electrónicas se optó por armar, en una placa universal, una placa complementaria que se coloca sobre el Arduino Uno y se conecta a ella mediante el acoplamiento de sus pines sin necesidad de alguna otra conexión externa. Este tipo de placa es normalmente denominada como "*Shield*"(Figura: 3.5). Esta misma contiene el *driver* del motor paso a paso y permite conectar rápidamente los sensores y el motor a través de pines diseñados especialmente para cada uno en el *Shield*.

³UART por su sigla en inglés *Universal Asynchronous Receiver-Transmitter*

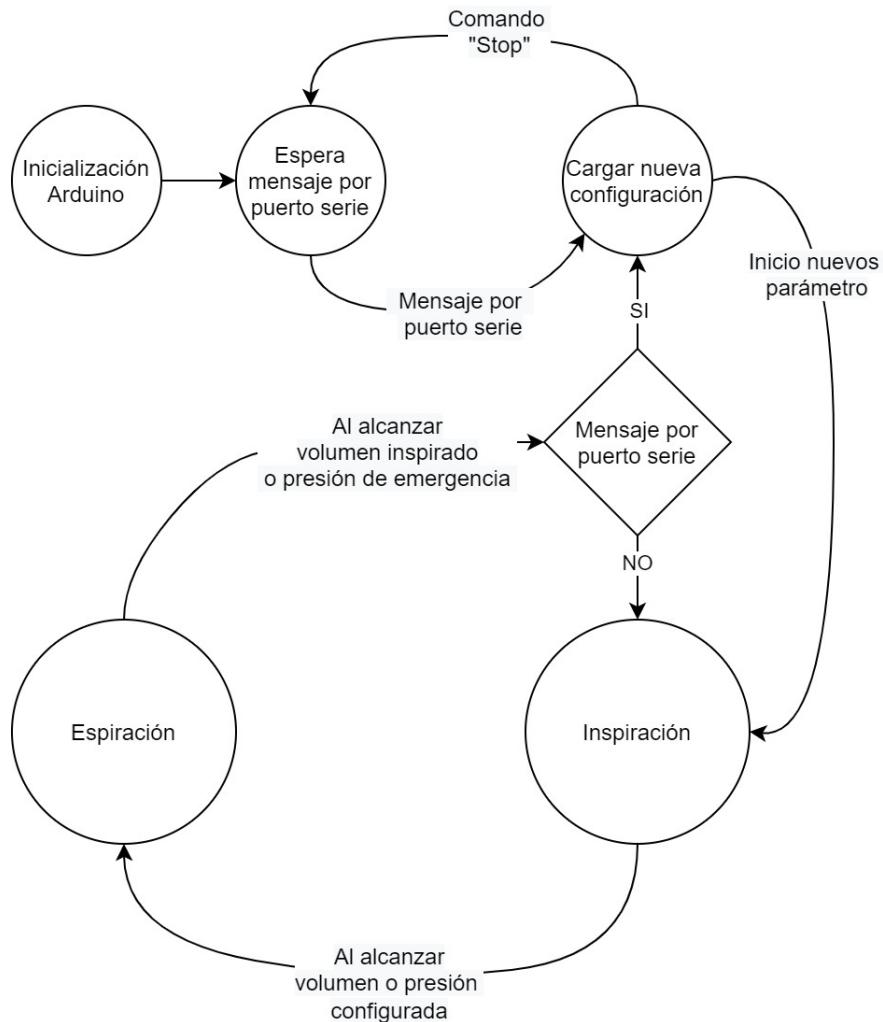


Figura 3.4: Diagrama de flujo, control de máquina de estados.

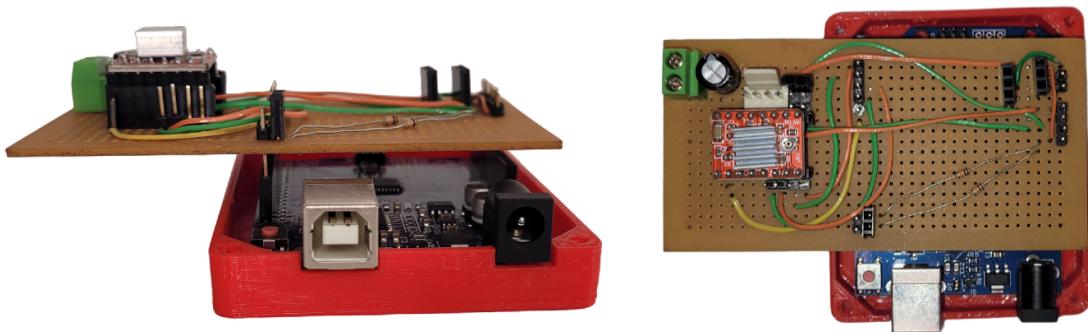


Figura 3.5: Imagen de la placa complementaria.

Placa complementaria desarrollada en placa universal, denominada normalmente como "Shield"

De acuerdo con la filosofía *Open Hardware* (sección 1.5.3), para que el equipo pueda ser replicado es necesario contar con una buena documentación del mismo. Con este objetivo se realizó un esquemático de todos los componentes (Figura: 3.6) y se diseñó un circuito impreso de la placa complementaria armada (Figura: 3.7). Estos mismos se encuentran publicados en un repositorio libre de git-hub⁴, para que cualquier persona interesada pueda acceder a ellos y utilizarlos para su propio proyecto.

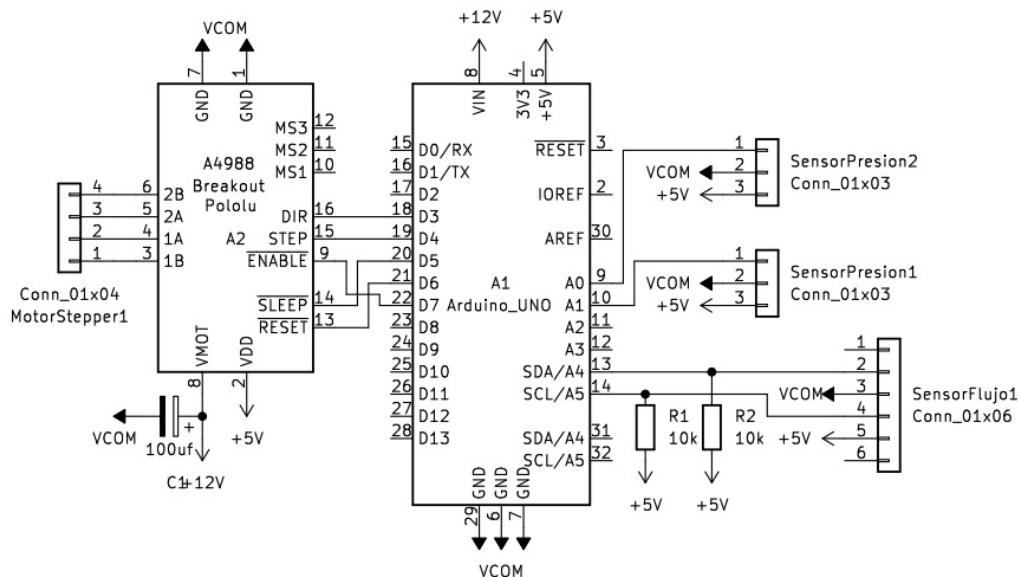


Figura 3.6: Esquemático del circuito electrónico.

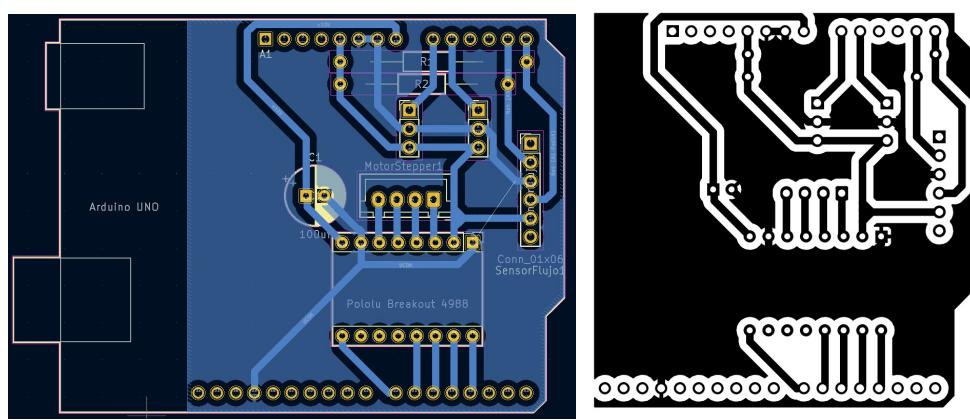


Figura 3.7: Diseño del circuito impreso de la placa *Shield* para Arduino Uno.

⁴<https://github.com/ifenoy/Lung-function>

El costo total de los insumos utilizados para armar el prototipo fue de U\$D 324,55. En este costo no se incluye la mano de obra del diseño y del armado. En la tabla 3.1 se muestra el costo individual de todos los componentes empleados para el desarrollo del prototipo. Como se puede observar en la misma, los sensores representan el 56 % del costo de desarrollo del prototipo, siendo el sensor de flujo el de mayor valor representando el 28 % del total .

Componentes y costos (U\$D)			
Insumo	Cantidad	Costo unitario	Costo Total
Acople flexible	1	13,50	13,50
Nema 17 39mm	1	26,60	26,60
Rodamiento 608 zz	4	9,00	36,00
Rodamiento lineal	4	2,00	8,00
Tornillo tipo Acme	1	9,30	9,30
Pack x50: Tuercas M3	1	1,50	1,50
Pack x50: Allen M3	1	7,50	7,50
Placa Arduino Uno	1	17,40	17,40
Rollo PLA 1.75mm	1	19,00	19,00
Placa universal	1	1,40	1,40
Placa Stepper Driver A4988	1	2,35	2,35
Sensor Flujo SFM3400-33-D	1	90,00	90,00
Sensor presión mpx5010dp	2	46,00	92,00
			Total: U\$D 324,55

Tabla 3.1: Tabla de costos.

3.2. Desarrollo de los programas de adquisición, visualización y procesamiento

Para garantizar la disponibilidad y accesibilidad del prototipo a la comunidad científica se optó por utilizar Python como lenguaje de programación, por ser este *open source* y ampliamente usado por esta comunidad. Se diseñaron dos programas, uno para el uso del prototipo durante las pruebas experimentales (Programa de Experimentación) y otro para el posterior procesamiento de los resultados obtenidos (Programa de Procesamiento).

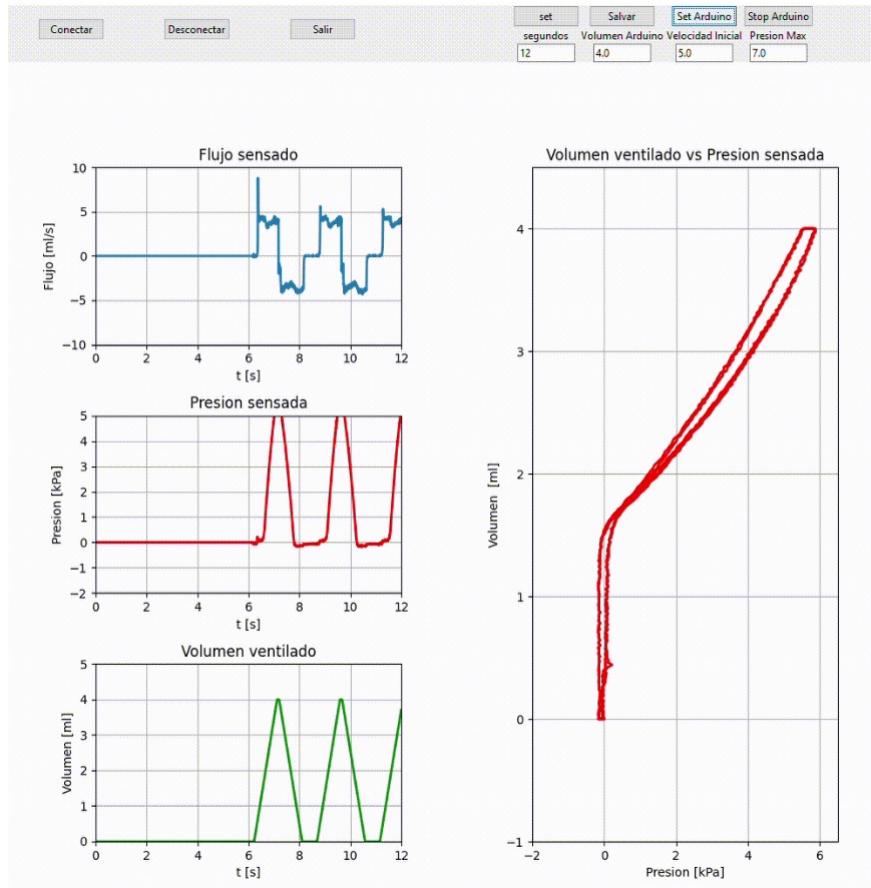


Figura 3.8: Interfaz gráfica del Programa de Experimentación.

Las mediciones se hicieron simulando el pulmón con un globo. Hipervínculo a una animación de la figura

El Programa de Experimentación se diseñó con el objetivo de permitir al usuario configurar los parámetros de ventilación y visualizar las mediciones en tiempo real. El mismo se comunica con el Arduino Uno a través del puerto UART y le envía las instrucciones de ventilación introducidas por el usuario. Estos parámetros incluyen el tiempo de muestreo, la velocidad de ventilación, el volumen máximo y la presión máxima. Además, este programa tiene la funcionalidad de mostrar gráficamente, en tiempo real y de forma animada, los valores medidos por los sensores que son recibidos desde el Arduino a través del mismo puerto UART. De esta manera, el usuario puede visualizar de manera clara y sencilla cómo se están registrando las mediciones y, en caso de ser necesario, puede cambiar los parámetros de ventilación de forma dinámica. Todo esto le permite al usuario evaluar si el ensayo se está realizando de manera correcta mientras se

adquieren los datos (en el Anexo Programa de Experimentación se describen las bibliotecas empleadas y el código del programa). Este programa también tiene la capacidad de almacenar las mediciones en un archivo CSV⁵. Esto es especialmente útil para su posterior procesamiento y análisis de forma correcta, ya que los datos se almacenan de forma estructurada y en un formato universalmente compatible.

En la figura 3.8 se muestra una captura de pantalla del programa, donde se observan los valores medidos en tiempo real y la interfaz de usuario para configurar los parámetros de ventilación.

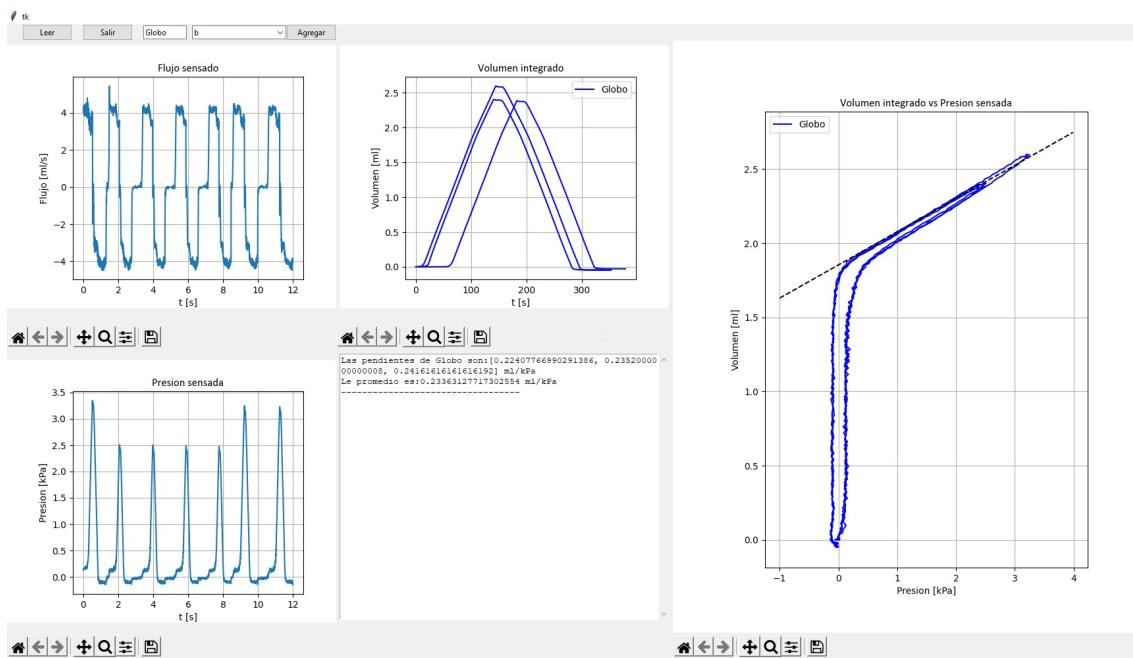


Figura 3.9: Interfaz gráfica del Programa de Procesamiento.

Las mediciones se hicieron simulando el pulmón con un globo.

Luego se diseñó el Programa de Procesamiento que permite analizar los datos que se almacenaron durante la experimentación. Para el desarrollo de este programa se utilizó una estrategia automatizada para procesar las mediciones obtenidas durante la ventilación. Dicho programa funciona de la siguiente manera: primero, se abre el archivo CSV que contiene las mediciones y

⁵CSV del inglés *Comma-Separated Values*

se carga en el programa. Luego, el programa de forma automática separa los ciclos de ventilación y calcula el volumen de cada ciclo, integrando el flujo de forma individual. Este proceso permite obtener una mayor precisión en el cálculo del volumen inspirado y espirado durante la ventilación. Además, calcula las pendientes de espiración para cada ventilación, obtiene el promedio y el desvío estándar de las mismas (en el Anexo Programa de Procesamiento se describen las bibliotecas empleadas y el código del programa).

La figura 3.9 muestra una vista del programa en funcionamiento, donde se pueden observar los gráficos de flujo y volumen de cada ciclo de ventilación, así como la gráfica de las pendientes de espiración. De esta forma, el usuario puede tener una visión general y detallada de los resultados obtenidos a partir de las mediciones realizadas con el prototipo.

Todo el código de ambos programas se encuentra publicado en el repositorio libre de Github <https://github.com/ifenoy/Lung-function>.

Capítulo 4

Calibración y validación del prototipo

Con el objetivo de validar el funcionamiento del prototipo desarrollado se procedió a calibrar la medición de los sensores. Luego de la calibración se buscó validar su funcionamiento con distintos modelos animales. A continuación se detallan estos procedimientos.

4.1. Calibración de los sensores de presión y flujo

4.1.1. Calibración del sensor de presión

Para validar las mediciones de presión del sistema, se llevó a cabo un proceso de calibración utilizando un calibrador patrón FLUKE 725 con módulo de presión FLUKE 700PD2. Se implementó un circuito de aire cerrado y se generaron 13 puntos de presión estática entre -6 y 6 kPa. A partir de estas mediciones se obtuvo el coeficiente que permite transformar la señal digital obtenida por el conversor analógico-digital del Arduino Uno a presión en kPa. Los resultados de la curva de calibración se pueden apreciar en la figura 4.1.

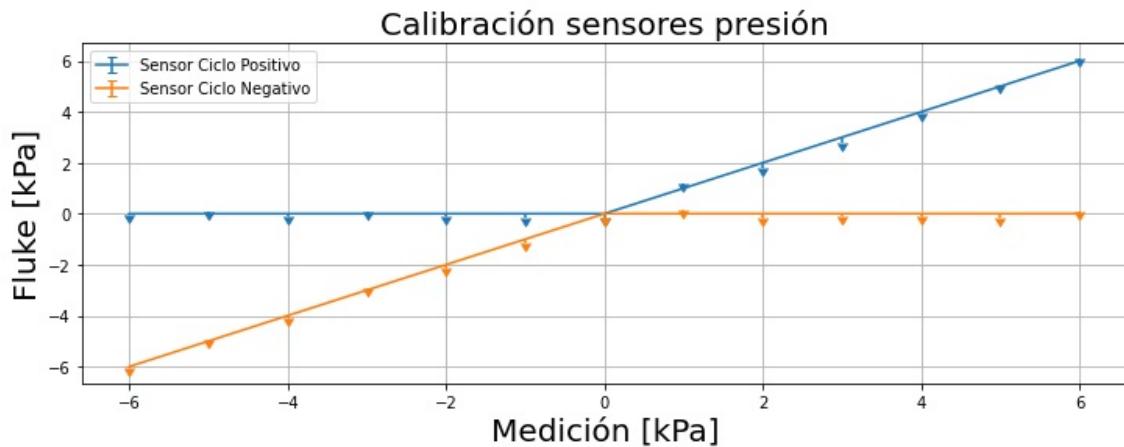


Figura 4.1: Mediciones de calibración de los sensores de presión.

En celeste se muestran las mediciones del sensor de presión que mide la presión positiva con respecto al calibrador patrón y en naranja las mediciones del sensor de presión que mida la presión negativa con respecto al calibrador patrón.

4.1.2. Verificación de pérdidas de presión

Durante la calibración del sistema, se realizó una prueba para asegurarse de que no hubiera pérdidas de presión en todo el circuito. Para ello, se colocó un globo en lugar de la cánula y se procedió a mover manualmente la bomba de jeringa hasta lograr una presión de 6 kPa. Esta presión se eligió porque supera a la presión máxima de funcionamiento experimental del sistema que es de 5 kPa. Se verificó que la medición no descendiera más de 5,94 kPa, que representa el 99 % del valor inicial, en menos de un minuto. Esta prueba demostró que el circuito estaba sellado y que no había fugas de presión.

4.1.3. Calibración del sensor de flujo

Para calibrar el sensor de flujo se realizaron 10 inyecciones manuales de aire a diferentes rangos de velocidades, utilizando una jeringa con un volumen constante de 1 ml. Luego, se ajustó el coeficiente del sensor para que la integral de la señal de flujo medida fuera igual a 1 ml. Para verificar la precisión de la calibración, se repitió el procedimiento 10 veces y se comprobó que el

error en el volumen fuera menor al 3 %, como se muestra en la figura 4.2.

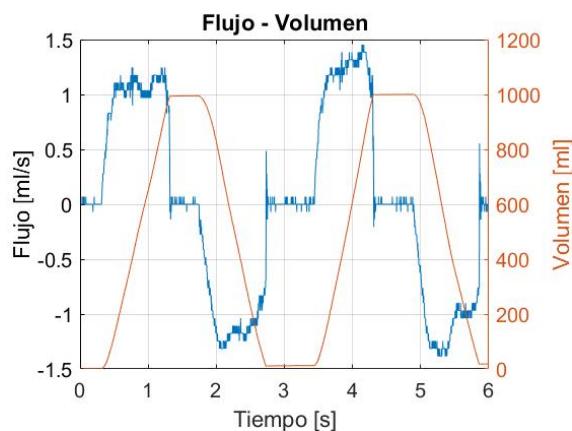


Figura 4.2: Mediciones para la calibración del sensor de flujo.

En celeste se muestran las mediciones del sensor flujo y en naranja el cálculo del volumen al integrar las mediciones del sensor.

4.2. Validación en animales

4.2.1. Condiciones de mantenimiento y cuidado de los animales de experimentación

Los ratones utilizados (25) a lo largo de este proyecto fueron comprados en el bioriego central de la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires. Los animales se mantuvieron en condiciones estándares, recibiendo agua y comida *ad-libitum* con un ciclo 12:12 horas de luz:oscuridad, humedad y temperatura ambiente controlada (22-24°C), en el bioriego del Centro de Estudios en Salud y Medio Ambiente de la Escuela de Ciencia y Tecnología perteneciente a la Universidad Nacional de San Martín (CESyMA-ECyT-UNSAM). Para el uso y cuidado de animales de laboratorio se siguieron las directivas del NIH¹, los lineamientos del Comité Institucional para el Cuidado y Uso de Animales de Experimentación² y la normativa de

¹NIH Publication - Guide for the Care and Use of Laboratory Animals: Eighth Edition

²CICUAE UNSAM 05/2022

referencia para bioterios de ANMAT³. La rata se compró al bioterio de la Facultad de Veterinaria de la Universidad de Buenos Aires y se mantuvo bajo las mismas condiciones que se detallaron previamente.

4.2.2. Procedimiento para medición en ratones

A los animales se les realizó la eutanasia mediante administración de isoflurano vía inhalatoria a través de una cabina saturada con el agente anestésico. Se les realizó una traqueostomía mediante entubado de la tráquea con un catéter de 18 G (BD), se conectó el catéter al prototipo (Figura: 4.3).

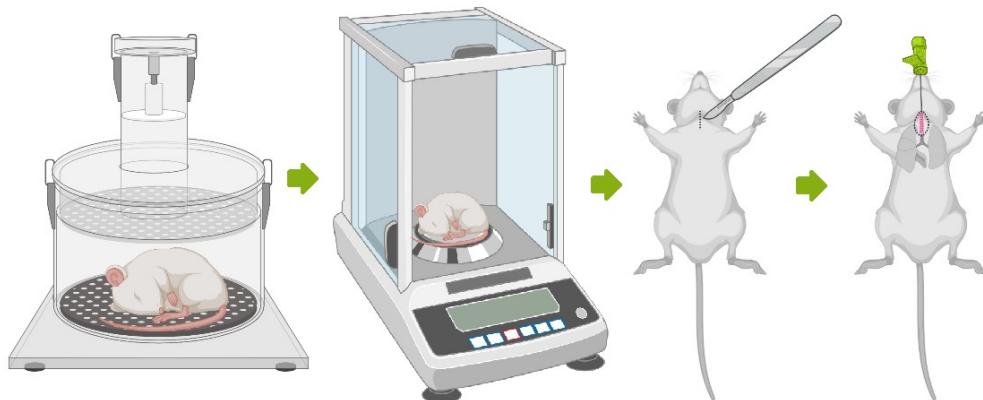


Figura 4.3: Proceso de eutanasia, pesaje y traqueostomía.
Proceso de eutanasia (izq), pesaje (centro) y traqueostomía (der).

A fin de disminuir la contribución de las vías aéreas superiores en la medición de la mecánica pulmonar, se buscó introducir la cánula lo más cerca posible de los pulmones teniendo cuidado de no obstruir el flujo de aire. La obstrucción es fácil de detectar puesto que durante el ciclo de expiración la presión medida tiende rápidamente a valores negativos (Figura: 4.4).

En el Programa de Experimentación se seleccionó una ventana de tiempo de 12 s, un flujo de 7 ml/s, un volumen máximo de 2,5 ml, una presión máxima de 5 kPa y se verificó que los

³Disposición N° 6344/96 de la Administración Nacional de Medicamentos, Alimentos y Tecnología Médica (ANMAT)

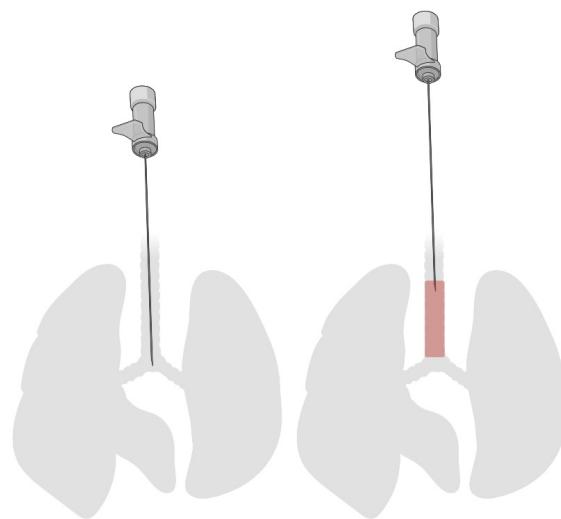


Figura 4.4: Posicionamiento del catéter en el pulmón,

En rojo está marcada la sección de las vías aéreas superiores que se busca minimizar.

sensores estén midiendo. Luego de la eutanasia los pulmones tienden a la atelectasia (colapso completo o parcial), por lo cual antes de comenzar a registrar las mediciones fue necesario realizar un reclutamiento alveolar mediante hiperinsuflación del pulmón. Estas maniobras se llevaron adelante hasta que se estabilizaron los bucles de presión-volumen. En ese momento se comenzó a registrar las mediciones y se verificó que la curva fuese la esperada.

Se graficó la medición del bucles presión-volumen de un ratón normal y se obtuvo una curva con una morfología esperable según la biografía [20] (Figura: 4.5). Además, se observó que los distintos bucles sucesivos medidos para un mismo ratón fueron muy similares entre sí. Esto último indicaría que la medición es muy confiable puesto que es estable para un mismo animal.

Al realizar las mediciones de los bucles presión-volumen de tres ratones normales de la misma camada se observó que las curvas obtenidas fueron muy similares entre sí (Figura: 4.6). Esto sugiere que el sistema de medición desarrollado es consistente y reproducible.

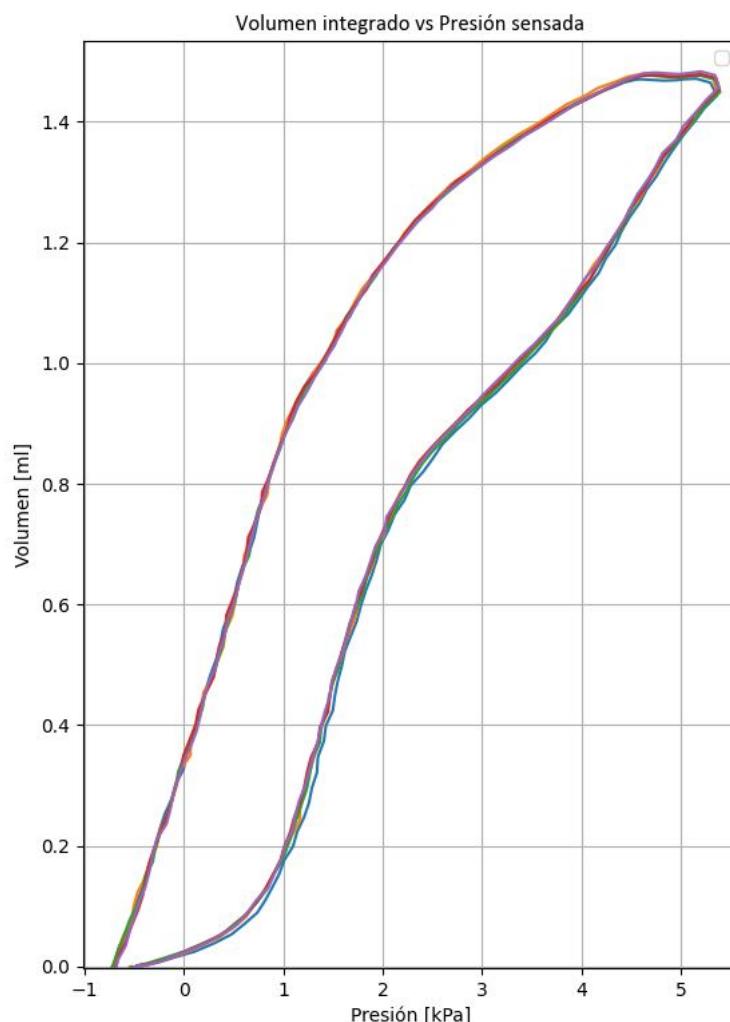


Figura 4.5: Mediciones representativas de bucles presión-volumen correspondientes a un ratón normal.

La *compliance* es una parámetro que se puede obtener de cualquier región linealizada de la curva PV no lineal. A fin de obtener un valor de *compliance* de los bucles obtenidos se decidió modelizar este parámetro a partir del tramo final espiratorio donde la relación entre estas dos magnitudes presenta un comportamiento lineal. Esta modelización fue previamente realizada por Limjunyawong et. al [12], quienes observaron que en el ratón, los extremos de deflación son lineales por debajo de 0,8 kPa y, por esta razón, es fácil definir una *compliance* reproducible en esta región. Entonces, para calcular la *compliance* (C), para cada bucle se realizó una recta tan-

gente a esta región final de la curva y a dicha recta se le calculó la pendiente. Para cada animal se promedieron las pendientes obtenidas de cada bucle. En la figura 4.6 A se representa una recta tangente para cada uno de los 3 animales. Al calcular la C para cada animal podemos observar que se hicieron 5 mediciones de pendientes y todas presentan valores similares que van desde 0,533 ml/kPa hasta 0,589 ml/kPa (Figura: 4.6 B), con un devío estándar de 0,007127, 0,02140 y 0,004134l, para N1, N2 y N3 respectivamente. Al comparar los promedios de C entre distintos animales se puede observar que presentan valores similares entre sí (figura: 4.6 B). Además, estos valores se encuentran en el rango de los previamente reportado para animales sanos [12]. En conjunto, todo esto indicaría que la parametrización de *compliance* realizada a partir de los datos obtenidos con nuestro dispositivo es robusta e informativa del comportamiento de la mecánica.

Otro parámetro informativo para la caracterización de la función pulmonar es la Capacidad Inspiratoria (IC). En humanos la IC se define por el esfuerzo voluntario máximo que una persona puede hacer. Desafortunadamente esto nunca se puede realizar en un modelo animal, por lo tanto, el volumen máximo en las curvas PV experimentales está determinado por una presión máxima establecida arbitrariamente por el investigador [12, 56]. En este trabajo, luego de muchas pruebas, decidimos trabajar a una presión máxima de 5 kPa, por lo que definimos la IC como el volumen alcanzado a esta presión máxima. En la figura 4.6 C, podemos observar los valores de volúmenes máximos alcanzados para cada bucle de cada ratón. Estos valores oscilan entre 1,43 ml y 1,51 ml, y presentan una desviación estándar de entre 0,02, 0,01 y 0,02 para N1, N2 y N3 respectivamente. El valor de IC se obtuvo al promediar todos los valores de volumen máximo para cada animal dando un valor de $1,45 \pm 0,01$ ml, $1,502 \pm 0,002$ ml y $1,48 \pm 0,01$ ml respectivamente. Podemos observar que estos valores fueron similares entre los 3 animales analizados.

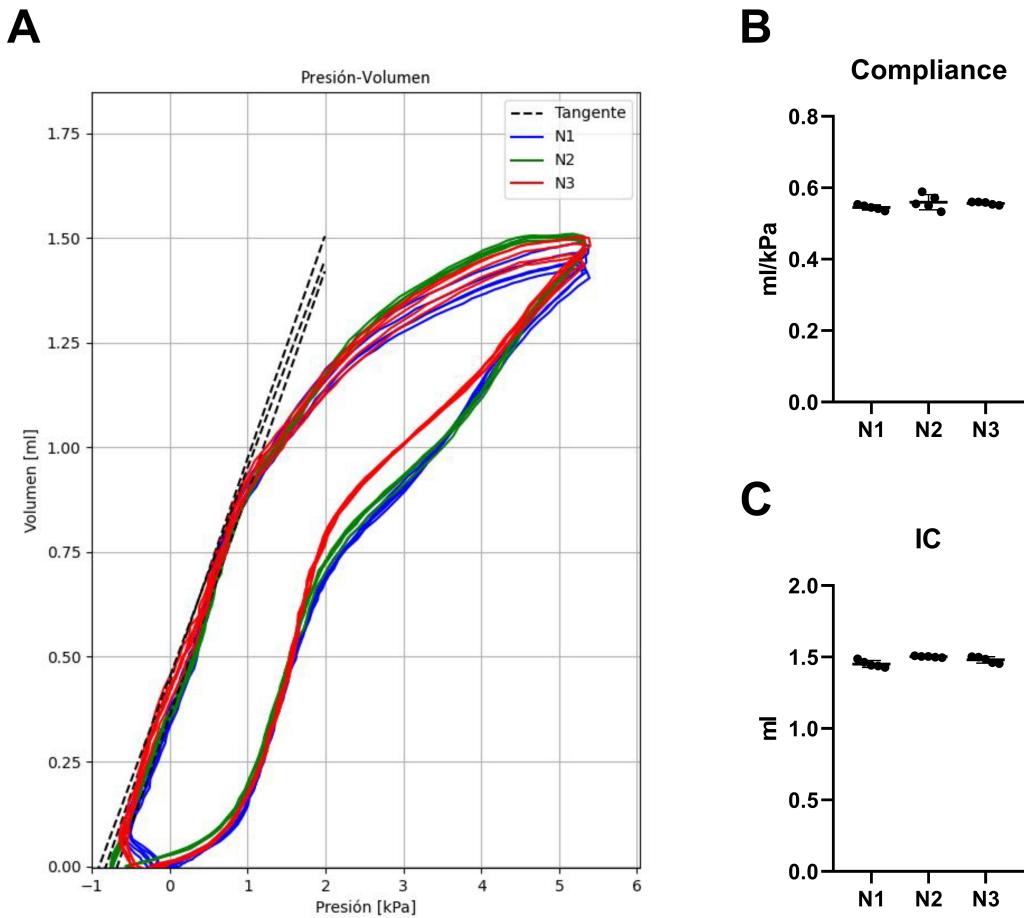


Figura 4.6: Mediciones representativas de bucles presión-volumen, compliance e IC correspondientes a tres ratón normales.

(A) Mediciones de bucles presión-volumen, (B) cálculo de compliance y (C) cálculo de IC.

La función pulmonar varía a lo largo de la vida y depende de la edad de los animales [56]. Se decidió entonces evaluar con el prototipo los bucles presión-volumen en ratones de distinta edad. A partir de la medición del bucle presión-volumen se calcularon la *compliance* y la Capacidad Inspiratoria en ratones de distintas edades (Figura: 4.7).

Se pudo observar que a medida que aumentaba la edad del ratón, también aumentaban la *compliance* y la Capacidad Inspiratoria. Los valores encontrados son similares a los reportados previamente por Elliott et al. [56]. Estos resultados pueden ser útiles para establecer comparacio-

nes en futuros estudios de la función pulmonar en ratones de diferentes edades y para contribuir al entendimiento de la fisiología pulmonar en diferentes etapas de la vida.

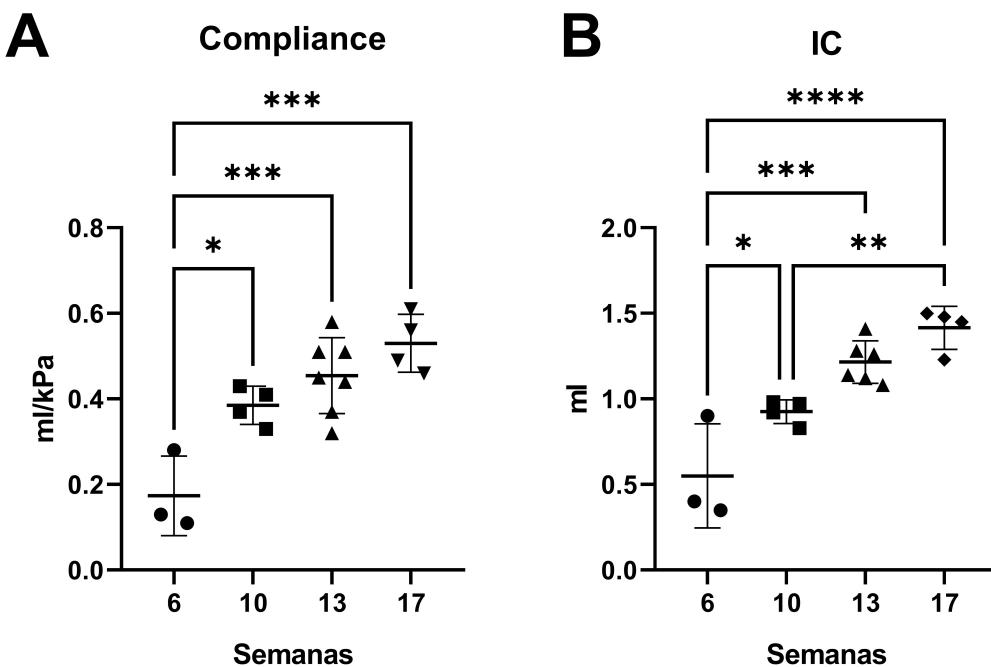


Figura 4.7: Compliance e IC correspondientes a ratones normales de diferentes semanas de edad. Cálculo de la compliance (izq) y de la Capacidad Inspiratoria (der). * $p<0,05$, ** $p<0,01$, *** $p<0,001$, **** $p<0,0001$, ANOVA⁴ con Tukey's test a posteriori.

4.3. Validación con modelos de patologías respiratorias

4.3.1. Modelo de asma agudo

Se empleó un modelo clásico de asma utilizando ovoalbúmina (OVA) como alérgeno y sensibilización parenteral (Figura: 4.8). Se seleccionó este modelo debido a su amplia caracterización en la literatura y dado que se emplea de rutina en nuestro laboratorio [57, 58, 59, 60]. Dicho modelo se basa en sensibilizar a los ratones mediante inmunización intraperitoneal (IP) con el alérgeno (OVA), en una dosis de 20 µg, junto con una solución de hidróxido de aluminio 2 mg

disuelta en PBS⁵. Las sensibilización se repite a la semana mediante otra administracion IP. Posteriormente, se realiza un desafío por vía aérea mediante la exposición a aerosoles de OVA al 3 % en PBS durante 20 minutos por 3 días consecutivos. La exposición por vía aérea se realizó en compartimientos individuales de una cámara conectada a un nebulizador (San-Up, San Martín, Argentina, flujo de la solución de OVA 0,33 ml/min en un flujo de aire de 6–8 l/min).

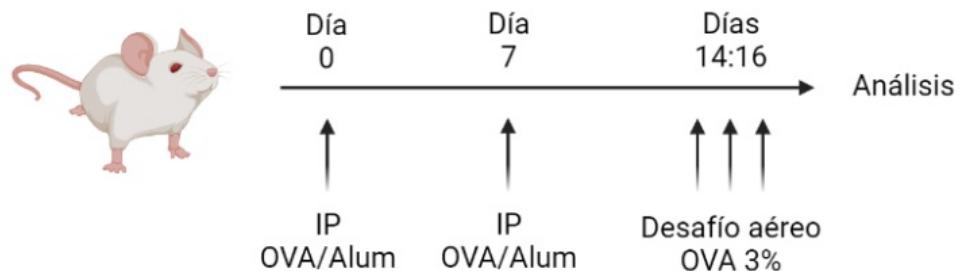


Figura 4.8: Modelo de asma agudo.

Modelo de asma utilizando ovoalbúmina (OVA/Alum) como alérgeno y sensibilización parenteral (IP) día 0 y 7. Desafío por vía aérea mediante la exposición a aerosoles de OVA al 3 % en PBS días 14, 15 y 16.

Histopatología e inflamación

El asma se caracteriza por una inflamación crónica de las vías aéreas con infiltrado celular en el pulmón e hiperplasia de las células productoras de mucus [61]. Para confirmar que los animales presentaban características de la patología, se analizó el efecto del modelo de asma en la histopatología mediante cortes histológicos del pulmón. Para esto los pulmones se fijaron por instilación con formalina⁶, se incluyeron en parafina y se cortaron con micrótomo. Las secciones transversales de pulmones se tiñeron con PAS⁷-Hematoxilina. El PAS tiñe los azúcares marcando de esta manera a las células productoras de mucus. Como se puede observar en la figura 4.9 los animales normales presentan vías áreas despejadas con estructura alveolar

⁵PBS por sus siglas en inglés *Phosphate Buffered Saline*

⁶Formol 40 tamponado al 10 % en PBS

⁷PAS por sus siglas en inglés *periodic acid schiff*

conservada. Al analizar la histología de animales sometidos al modelo de asma, se observa infiltrado celular peribronquial y perivascular. Además se puede observar hipertrofia e hiperplasia de células productoras de mucus en los bronquiolos (Figura: 4.9).

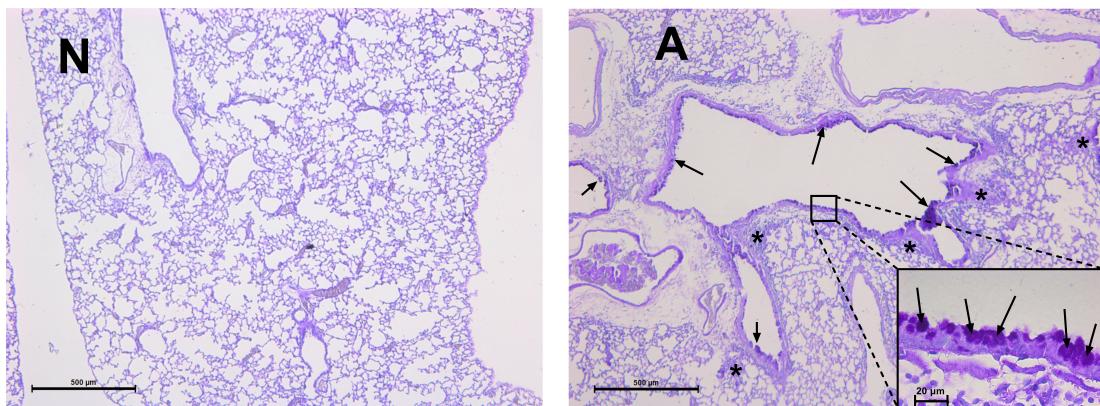


Figura 4.9: Histología de pulmón (tinción con PAS-Hematoxilina).

Los asteriscos indican infiltrado celular y las flechas indican células productoras de mucus. Se muestra una fotografía de una muestra representativa de ratón normal (N) y del modelo de asma (A).

Una vez confirmado el fenotipo asmático a través de la histopatología, se llevaron a cabo mediciones de los bucles de presión-volumen en ratones sanos y en animales a los que se les realizó el modelo de asma agudo. Se obtuvieron curvas con una morfología esperada según la bibliografía [17] (Figura: 4.10). En el grupo de ratones asmáticos se observó que a iguales presiones de trabajo se alcanzó un menor valor de volumen máximo. Asimismo, al analizar la región final de espiración se observó una menor pendiente en los animales asmáticos en comparación con los ratones sanos. Esto sugeriría que, como indica la literatura del área, el asma dismuye el valor de *compliance* [17].

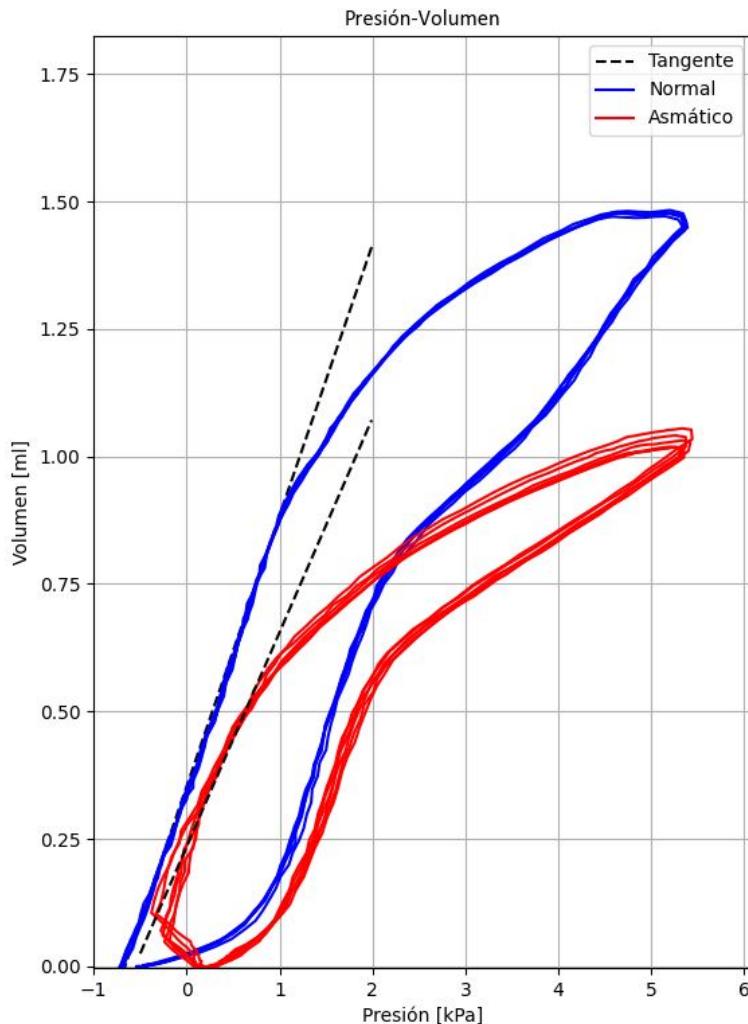


Figura 4.10: Mediciones representativas de bucles presión-volumen correspondientes a un ratón normal y a un ratón del modelo de asma agudo.

Mediciones representativas de bucles presión-volumen correspondientes a un ratón normal (azul) y a un ratón al que se le realizó el modelo de asma agudo (rojo). Las líneas punteadas corresponden a las rectas tangentes del último tramo espiratorio.

Al comparar los resultados de los cálculos de *compliance* e IC en los ratones sanos y los modelos de asma agudo, se pudo observar una disminución significativa en el grupo asmático respecto de los animales normales (Figura: 4.11). Esta diferencia significativa se observó mediante la realización de un análisis estadístico ($p<0,05$ Unpaired T-Test). Esto indica que el prototipo es capaz de detectar las diferencias en la función pulmonar entre ratones sanos y asmáticos.

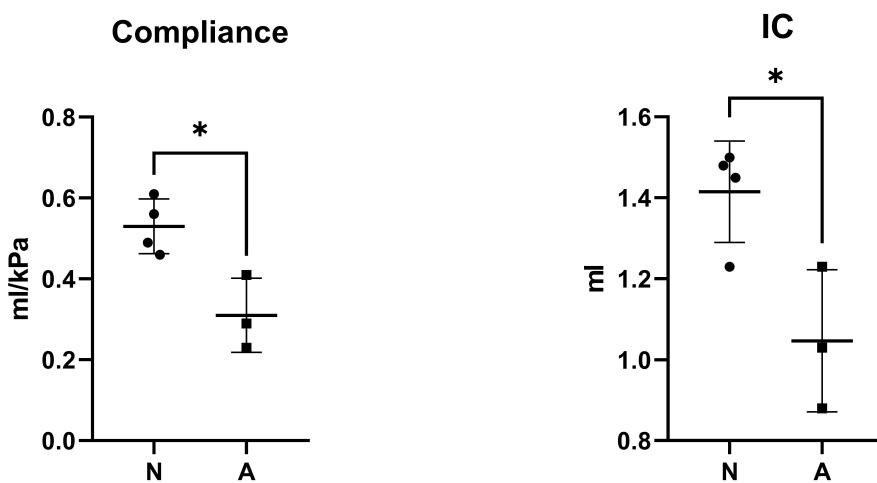


Figura 4.11: Compliance e IC correspondientes a ratones normales y a ratones del modelo de asma agudo.

Cálculo de la compliance y de la Capacidad Inspiratoria correspondientes a ratones normales (N) y a ratones a los que se les realizó el modelo de asma agudo (A).

4.3.2. Modelo de asma crónico

Se empleó un modelo de asma crónico, que es equivalente al asma agudo pero posea más desafíos aéreos. El número de desafíos aumenta progresivamente, con dos días en la primera semana, y tres días en la segunda y tercera semana.



Figura 4.12: Modelo de asma crónico.

Modelo de asma utilizando ovoalbúmina (OVA/Alum) como alérgeno y sensibilización parenteral (IP) día 0 y 7. Desafío por vía aérea mediante la exposición a aerosoles de OVA al 3% en PBS días 14, 15, 21, 22, 23, 28, 29 y 30.

Se realizaron mediciones de los bucles de presión-volumen en ratones con un modelo de asma crónico y se encontraron resultados significativamente diferentes en comparación con los ratones sanos (Figura: 4.13). Como era de esperar, puesto que es un modelo más severo, la curva del bucle en los animales asmáticos alcanzan menores valores de volumen respecto a los animales normales, presentando una menor pendiente en el segmento final de la curva de espiración. Esto indica una disminución en la capacidad pulmonar y una mayor disnea.

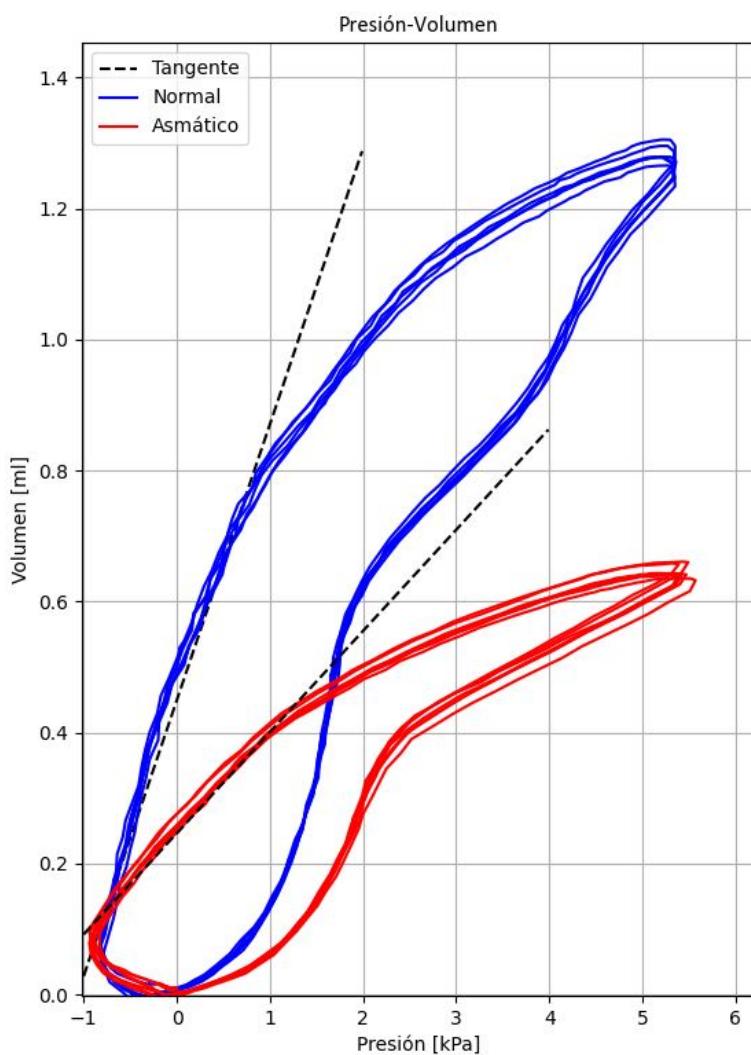


Figura 4.13: Mediciones representativas de bucles presión-volumen correspondientes a un ratón normal y un ratón del modelo de asma crónico.

Mediciones correspondientes en azul a un ratón normal y en rojo a un ratón al que se le realizó el modelo de asma crónico. Las líneas punteadas corresponden a las rectas tangentes del último tramo espiratorio.

Además, al calcular la *compliance* y el IC, se encontró que ambos valores fueron significativamente menores en el modelo de asma crónico en comparación con los ratones sanos ($p<0,01$ *Unpaired T-Test*) (figura: 4.14). Esto indica que el prototipo es capaz de detectar de manera precisa y confiable las diferencias en la función pulmonar en varios modelos de asma.

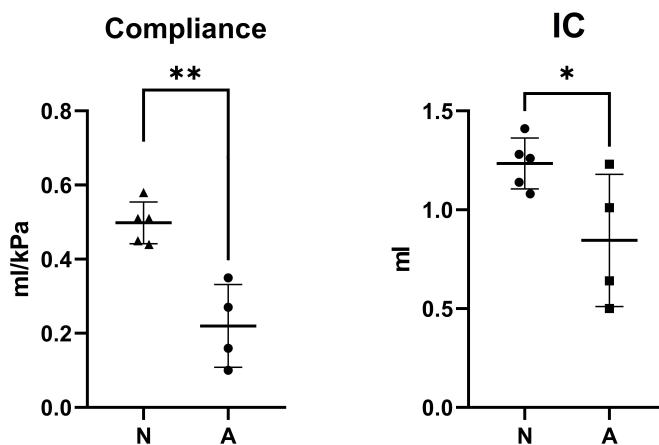


Figura 4.14: Compliance e IC correspondientes a ratones normales y a ratones del modelo de asma crónico.

Cálculo de la *compliance* (A) y de la Capacidad Inspiratoria (B) correspondientes a ratones normales (N) y a ratones a los que se les realizó el modelo de asma crónico (A).

4.4. Validación con otro modelo animal de experimentación

4.4.1. Procedimiento para medición en rata

Con el objetivo de evaluar la capacidad del equipo, se llevó a cabo la medición del bucle presión-volumen en una rata. Se utilizó el mismo procedimiento que se empleó en ratones, tal como se describe en la sección 4.2.2, con la diferencia del tamaño del catéter utilizado. Dado el mayor tamaño de la tráquea del animal en comparación con la de los ratones, se optó por utilizar un catéter de 14 G en lugar de uno de 16 G.

Con el dispositivo desarrollado, fue posible adquirir las señales de presión y flujo necesarias para la medición del bucle presión-volumen en ratas, tal como se muestra en la Figura 4.15 A. Se puede observar que trabajando a las mismas presiones que las que empleadas para ratón, se alcanzaron mayores valores de volumen. Esto hace que todo el bucle sea más empinado, presentando así una mayor pendiente en comparación a los obtenidos para ratón.

Al calcular el valor de *compliance*, se observa que se obtuvo un valor medio de $4,61 \pm 0,05$ ml/kPa (Figura: 4.15 B). Este valor está en el orden del reportado previamente por Gaultier et al. [62]. Por otro lado, al evaluar la Capacidad Inspiratoria para la rata, observamos un valor medio de $9,22 \pm 0,02$ ml (Figura: 4.15 C).

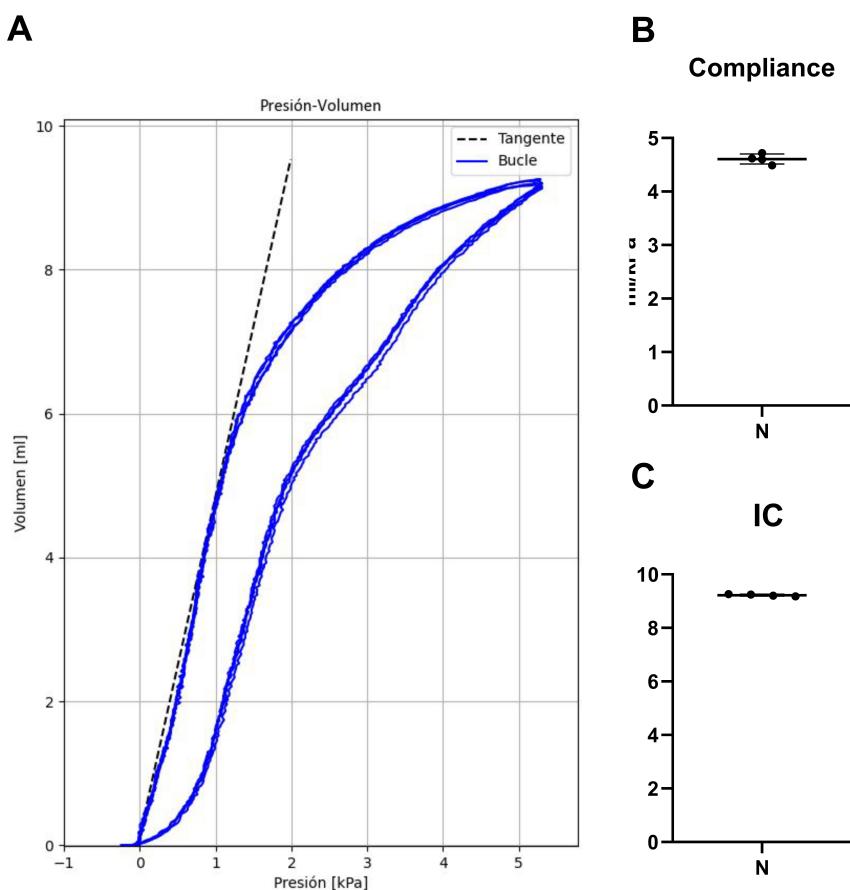


Figura 4.15: Mediciones representativas de bucle presión-volumen de una rata.
(A) Mediciones de bucles presión-volumen, (B) cálculo de *compliance* y (C) Cálculo de IC.

Estos resultados en su conjunto demuestra la efectividad del prototipo y su versatilidad para medir la función pulmonar en diferentes modelos animales que se emplean comúnmente en ensayos pre-clínicos.

Capítulo 5

Discusión

Las enfermedades respiratorias crónicas son una de las principales causas de morbilidad a nivel mundial [1]. Existe una urgente necesidad de estudiar los mecanismos subyacentes a estas patologías así como de desarrollar nuevas terapias efectivas contra las mismas. Las pruebas de función pulmonar son herramientas importantes para describir las características fenotípicas de una enfermedad respiratoria, pero el desarrollo de estas pruebas ha sido un gran desafío en animales pequeños de experimentación [6].

En este proyecto se desarrolló un prototipo de sistema *open source* capaz de medir los bucles de presión-volumen pulmonar en murinos. Este sistema, basado en una ventilación mecánica a circuito cerrado de aire, utiliza sensores comerciales de presión y flujo y, mediante el cálculo de la *compliance* y la Capacidad Inspiratoria (IC), permite diferenciar patologías pulmonares. Este sistema se validó utilizando dos modelos de asma, demostrando su capacidad para distinguir entre diferentes condiciones pulmonares. El diseño de código abierto del prototipo permite su replicación y adaptación a diferentes entornos de investigación. Y se espera que sea una herramienta útil para la comunidad científica y que permita democratizar el estudio de enfermedades respiratorias.

Se validó el prototipo realizando las primeras mediciones en animales. En primer lugar se buscó determinar los parámetros de trabajo. Para definir el flujo de ventilación con el que se tra-

bajaría se varió el mismo entre 5 y 10 ml/s. Se optó por 7 ml/s puesto que era el menor valor de flujo en el cual el registro era constante, sin que se aprecie en la medición de flujo los pasos discretos propios de la bomba paso a paso. Se buscó trabajar al menor flujo posible para evitar que se genere turbulencia. Es importante mencionar que el sensor de flujo utilizado no mide el flujo turbulento, por lo que se tomó en cuenta esta consideración teórica para el diseño y desarrollo del prototipo. En este sentido, cabe aclarar que si bien no se evaluó la presencia de flujo turbulento, se puede asumir que, dada la anatomía propia del pulmón de los ratones en particular por los diámetros pequeños de las vías respiratorias, el equipo trabaja con flujos lineales [34].

Seguidamente se determinó la presión máxima de trabajo. Para esto se evaluaron distintas presiones máximas entre 2 y 7 kPa. A presiones mayores a 7 kPa se observaba daño pulmonar por barotrauma. Se optó por la presión de 5 kPa que era un valor intermedio a los ensayados y dado que se observaban bucles presión-volumen similares a los que se esperan por la bibliografía [20]. Además, trabajando a esta presión se obtuvieron segmentos de los bucles presión-volumen muy lineales desde donde se pudo parametrizar la *compliance* fácilmente. Cabe aclarar que distintos grupos de investigación emplean diferentes valores de presión máxima [56]. Sin embargo, todos se encuentran dentro del rango de presiones de funcionamiento del prototipo desarrollado, por lo que podría adaptarse a los diferentes requerimientos de los usuarios funcionando dentro del abanico de presiones reportadas en la bibliografía [12, 20, 56]. Finalmente, los últimos parámetros de funcionamiento empleados fueron la presión mínima negativa y el volumen máximo. Ambos se eligieron por cuestiones de seguridad para evitar el daño de la estructura pulmonar.

Una vez establecidos los parámetros de trabajo se realizaron las mediciones definitivas en los distintos modelos experimentales. Muchos parámetros informativos de la capacidad pulmonar, tales como FVC, IC y VC, se definen en humanos realizando respiración forzada. Lamentablemente, esta prueba no puedo reproducirse en animales de experimentación y estos volúmenes se calculan a una presión arbitraria. Para nuestro cálculo de IC se eligió el valor de 5 kPa. Los ratones BALB/c de seis semanas mostraron un valor de la IC de $0,55 \pm 0,2$ ml, mientras que ratones de 29 semanas presentaron un valor de $1,32 \pm 0,02$ ml. En este sentido, el grupo de Mitz-

ner [20] trabajando a presión máxima de 3 kPa con ratones BALB/c de 10 semanas reportó una IC de $1\pm0,1$ ml. Por otro lado, el grupo de Sieck [56] trabajando también a 3 kPa de presión reportó en ratones C57BL/6 un volumen máximo de $0,61\pm0,03$ ml a las 8 semanas de edad y de $0,83\pm0,02$ ml a los 24 semanas de edad. Estos reportes indicarían que a pesar de las diferencias de edad, cepas y presión de trabajo, nos encontramos en el mismo orden de la IC medido.

Al evaluar la mecánica pulmonar de ratones asmáticos observamos que en ambos modelos se ve una reducción de al menos el 50 % de la *compliance*. Esta observación se encuentra en línea con lo reportado previamente por Kalidhindi *et al.* [63] en animales sensibilizados. Además, se observaron diferencias en el valor de la IC registrando una reducción en los animales asmáticos de más del 25 % respecto a sus respectivos controles. Contrariamente a los resultados reportados en el presente informe, Devos *et al.* [64] no reportaron cambios en este parámetro en animales sensibilizados. Esta diferencia podría deberse a que los autores emplearon un modelo de asma leve con sensibilización y desafío por vía intranasal sin el empleo de adyuvantes como en nuestro modelo. Por otro lado, no nos fué posible comparar animales asmáticos agudos y crónicos entre sí puesto que eran camadas diferentes de distinta edad.

Como validación final se decidió probar si el equipo era capaz de evaluar la mecánica pulmonar de una rata. Haciendo leves modificaciones, como variar el calibre del catéter empleado para realizar la traqueostomía, fue posible adquirir mediciones de curvas presión-volumen con morfología correcta de acuerdo con la bibliografía [62]. Para esta prueba de concepto se empleó solo un animal simplemente para tener idea si el equipo era capaz de funcionar en estas condiciones. Para obtener valores de interés fisiológico se deben realizar ensayos con un número mayor de animales. Esta validación en una especie con la que no se había trabajado hasta el momento da cuenta de la versatilidad del prototipo alcanzado y al mismo tiempo abre la puerta para ser probado en otros modelos animales pequeños de experimentación empleados para estudiar enfermedades respiratorias, ya sean roedores como son el caso de los hamsters o los cobayos [65, 66], e incluso primates pequeños como *Callithrix jacchus* [67].

Métodos de medición de la función pulmonar

Se han descripto diferentes maneras de evaluar la función pulmonar en animales pequeños de experimentación [10, 12, 38, 39]. En este sentido, existen diferentes métodos de medición disponible, tanto invasivos como no invasivos, pero debido a que ninguno de estos métodos resulta ideal en todos sus aspectos [68], determinar la mejor manera de evaluar la función pulmonar para una determinada enfermedad respiratoria sigue siendo un tema de controversia y debate [6]. Entre los métodos no invasivos se destacan por su amplio uso aquellos que emplean plethysmografos de cuerpo entero. Sin embargo, muchos investigadores han criticado su uso, en particular por el parámetro Penh que se obtiene por estos métodos por ser defectuoso para medir la mecánica de las vías respiratorias [41, 69, 70]. En este sentido Bates *et al.* [69] demostraron que el parámetro empírico Penh no es un estimador preciso de la resistencia de las vías respiratorias (Rn) y que debe complementarse con medidas más invasivas de la mecánica pulmonar [69]. Estos abordajes invasivos se realizan con maniobras de ventilación forzada en animales anestesiados y traqueostomizados [40]. Como mencionamos en la introducción, actualmente existen dos dispositivos comerciales que emplean métodos invasivos que están disponibles para medir directamente la función pulmonar. Ambas tecnologías miden las curvas de presión-volumen, lo que da como resultado parámetros clínicamente relevantes, como la IC, la resistencia y la *compliance*. Es por esto que para el desarrollo de este prototipo se eligió un método invasivo para evaluar la función pulmonar porque brinda mediciones más confiables y tiene una tecnología más sencilla y accesible de replicar.

Dificultades encontradas

Como se mencionó previamente, el dispositivo descripto en el presente informe es el resultado de un proceso iterativo en ciclos de definición de requisitos, diseño y desarrollo de un prototipo, experimentación con animales, evaluación de los resultados y retroalimentación. Estos ciclos dieron lugar a aprendizajes que moldearon al prototipo final. Consideramos interesan-

te presentar algunos de estos aprendizajes para discutir las características y limitaciones del prototipo final.

Inicialmente se evaluó la función pulmonar con animales anestesiados sin ventilación asistida. Para esto desarrollamos un prototipo que funcionaba a circuito abierto de aire para medir presión y flujo respiratorio. Sin embargo, la diferencia de presiones medidas en esta configuración resultaba similar al error de medición del sensor, por lo que la señal tenía una magnitud similar al ruido. Como entre nuestros objetivos estaba generar un equipo de fácil acceso para que pueda ser reproducido por otros grupos de investigación, se optó por seguir usando sensores comerciales. Para esto, decidimos trabajar a circuito cerrado de aire con ventilación manual. Esta modificación permitió aumentar las diferencias de presiones y que la presión medida en la tráquea fuese lo más parecida a la presión intrapulmonar.

Al realizar las primeras mediciones a circuito cerrado de aire con ventilación manual, empleando animales anestesiados, encontramos que era difícil sincronizar la ventilación normal de los ratones con la ventilación manual. Esto generaba que ambas ventilaciones se vean reflejadas en las mediciones de manera desfasada, lo cual dificultaba la caracterización de la mecánica pulmonar. Por esto se decidió entonces trabajar con animales luego del proceso de eutanasia. En este sentido, cabe aclarar que todos los métodos invasivos de medición de la función pulmonar que se encuentran comercialmente son terminales. Se realizó la eutanasia de los animales inmediatamente finalizada la medición.

Cuando se emplearon ratones luego de la eutanasia, se identificó que el uso de la ventilación manual para realizar mediciones en modelos animales requería una gran habilidad técnica del operador. Esto generaba problemas repetibilidad en las mediciones de un mismo animal y entre animales. Para salvar esta dificultad, decidimos adaptar una bomba de infusión de jeringa *open source* para que se realice la ventilación de manera mecánica. De esta forma se logró que el dispositivo sea menos operador-dependiente.

Todos estos dispositivos iniciales poseían un único sensor de presión con la idea de sim-

plificar el prototipo y reducir los costos midiendo sólo la región positiva de la curva. Sin embargo, luego de realizar varios ensayos se observó que una fracción considerable del bucle presión-volumen quedaba en el extremo negativo por lo que quizás se perdía parte informativa de la curva. Al no tener los bucles completos se dificultaba el análisis global de la morfología de estos y al mismo tiempo se podía ver afectada la medición precisa de la *compliance*. Para superar este problema, se decidió incorporar un segundo sensor de presión para medir por completo la recta de expiración incluyendo los valores negativos de presión. Se conectaron dos sensores de presión diferenciales, uno midiendo la presión positiva y el otro la negativa. De esta manera, se lograron obtener los bucles completos presión-volumen y mediciones más precisas y confiables de la *compliance* pulmonar.

Filosofía open hardware

La simplicidad del diseño presentado en este trabajo se ha buscado de forma deliberada, utilizando exclusivamente componentes comerciales o de fácil ensamblado que aprovechan las ventajas de la impresión 3D y la filosofía *open source*. Esto ha permitido crear un prototipo altamente replicable con un costo de desarrollo que representa aproximadamente el 5% o menos de un equipo comercial con características y rendimiento similares. Además, la flexibilidad del diseño permite una personalización virtualmente infinita para adaptarlo a los objetivos específicos de cada investigación, eliminando la necesidad de conformarse con las opciones disponibles en el mercado.

Con esto en mente y con el deseo de que este desarrollo promueva la investigación de patologías respiratorias dentro de la comunidad científica, se han realizado presentaciones en diferentes congresos y hemos subimos toda la información a un repositorio bajo una licencia libre.

Desafíos futuros

Para realizar una validación final aún más exhaustiva del prototipo hubiera sido interesante contar con un equipo comercial y contrastar los valores obtenidos en nuestro prototipo respecto al comercial empleando los mismos animales. No obstante, la gran reproducibilidad y robustez en las mediciones de nuestro equipo, junto con las curvas y valores obtenidos que se encuentran en el orden de lo esperado según la bibliografía, nos llevan a concluir que se ha logrado validar ampliamente el prototipo obtenido.

Por otro lado, si bien en el presente trabajo se han modelado algunos parámetros informativos de las curvas presión-volumen, esperamos que futuros investigadores modelen otros parámetros como son Resistencia, TLC, *Compliance Específica*, Volumen Residual, entre otros de acuerdo a la patología y la pregunta científica que deseen estudiar.

Capítulo 6

Conclusiones

Después de haber realizado el trabajo final integrador y haber cumplido con los objetivos planteados, podemos concluir que hemos logrado desarrollar un prototipo *open source* que permite obtener, mediante un método invasivo, las curvas de presión-volumen en ratones de laboratorio. Dichas curvas nos han permitido evaluar la función pulmonar mediante el cálculo de la *compliance* y de la IC en animales pequeños de experimentación. También hemos desarrollado un entorno gráfico que nos ha permitido visualizar en tiempo real las curvas obtenidas, lo que ha sido de gran utilidad para la validación del prototipo. Asimismo, durante el proceso de desarrollo, hemos implementado un algoritmo que permite calcular los parámetros de *compliance* e IC a partir de los datos obtenidos. Este algoritmo ha sido fundamental para la evaluación de la función pulmonar y consecuente validación del prototipo.

Finalmente, hemos validado el prototipo utilizando distintos modelos animales. Por un lado, empleando patologías respiratorias en ratones, lo que ha permitido comprobar su eficacia y utilidad en la investigación de enfermedades respiratorias. Por otro lado, empleando distintas especies animales, lo que muestra la gran versatilidad del prototipo alcanzado.

En resumen, podemos concluir que el prototipo desarrollado tiene un gran potencial como herramienta para democratizar la investigación y el desarrollo en el área de las enfermedades respiratorias, siendo sencillo y de bajo costo, lo que permite que pueda ser replicado por otros

grupos de investigación.

Bibliografía

- [1] Joan B. Soriano et al. «Prevalence and attributable health burden of chronic respiratory diseases, 1990–2017: a systematic analysis for the Global Burden of Disease Study 2017». En: *The Lancet Respiratory Medicine* 8 (6 jun. de 2020), págs. 585-596. issn: 2213-2600.
- [2] Organización Panamericana de la Salud. «OPS. La carga de las enfermedades respiratorias crónicas en la Región de las Américas, 2000-2019.» En: (2021). url: <https://www.paho.org/es/enlace/carga-enfermedades-respiratorias-cronicas>.
- [3] Stephen T. Holgate et al. «The Future of Asthma Care: Personalized Asthma Treatment». En: *Clinics in Chest Medicine* 40 (mar. de 2019), págs. 227-241.
- [4] David M. Mannino y A. Sonia Buist. «Global burden of COPD: risk factors, prevalence, and future trends». En: *The Lancet* 370 (sep. de 2007), págs. 765-773.
- [5] David J Lederer y Fernando J Martinez. «Idiopathic pulmonary fibrosis». En: *New England Journal of Medicine* 378.19 (2018), págs. 1811-1823.
- [6] Jeroen AJ Vanoirbeek et al. «Noninvasive and invasive pulmonary function in mouse models of obstructive and restrictive respiratory diseases». En: *American journal of respiratory cell and molecular biology* 42.1 (2010), págs. 96-104.
- [7] Charles E Reed. *Asthma and Rhinitis*. John Wiley & Sons, 2008.
- [8] A. Vinegar, E. E. Sinnett y D. E. Leith. «Dynamic mechanisms determine functional residual capacity in mice, *Mus musculus*». En: *Journal of Applied Physiology* 46 (1979), págs. 867-871.
- [9] Jason HT Bates et al. «Oscillation mechanics of the respiratory system». En: *Comprehensive Physiology* 1.3 (2011), págs. 1233-1272.
- [10] Charles G Irvin y Jason HT Bates. «Measuring the lung function in the mouse: the challenge of size». En: *Respiratory research* 4 (2003), págs. 1-9.
- [11] Charles G Irvin, Juno Pak y Richard J Martin. «Airway–parenchyma uncoupling in nocturnal asthma». En: *American journal of respiratory and critical care medicine* 161.1 (2000), págs. 50-56.

- [12] Nathachit Limjyunawong et al. «Measurement of the pressure-volume curve in mouse lungs». En: *JoVE (Journal of Visualized Experiments)* 95 (2015), e52376.
- [13] Jason H.T. Bates. «Mechanical Properties of the Lung». En: *Comparative Biology of the Normal Lung: Second Edition* (ene. de 2015), págs. 289-304.
- [14] Masaki Fujita et al. «Overexpression of tumor necrosis factor- α produces an increase in lung volumes and pulmonary hypertension». En: *American Journal of Physiology - Lung Cellular and Molecular Physiology* 280 (2001).
- [15] David A Kaminsky et al. «Hyperpnea-induced changes in parenchymal lung mechanics in normal subjects and in asthmatics». En: *American journal of respiratory and critical care medicine* 155.4 (1997), págs. 1260-1266.
- [16] D Papandrinopoulou, V Tzouda y G Tsoukalas. «Lung compliance and chronic obstructive pulmonary disease». En: *Pulmonary medicine* 2012 (2012).
- [17] PW Holmes, AH Campbell y CE Barter. «Acute changes of lung volumes and lung mechanics in asthma and in normal subjects». En: *Thorax* 33.3 (1978), págs. 394-400.
- [18] HJ Colebatch, KE Finucane y MM Smith. «Pulmonary conductance and elastic recoil relationships in asthma and emphysema». En: *Journal of Applied Physiology* 34.2 (1973), págs. 143-153.
- [19] KE Finucane y HJ Colebatch. «Elastic behavior of the lung in patients with airway obstruction». En: *Journal of applied physiology* 26.3 (1969), págs. 330-338.
- [20] Annette Robichaud et al. «Automated full-range pressure-volume curves in mice and rats». En: *Journal of Applied Physiology* 123.4 (2017), págs. 746-756.
- [21] Karen L Steinmetz y Edward G Spack. «The basics of preclinical drug development for neurodegenerative disease indications». En: *BMC neurology* 9.1 (2009), págs. 1-13.
- [22] P Mukherjee et al. «Role of animal models in biomedical research: a review». En: *Laboratory Animal Research* 38.1 (2022), pág. 18.
- [23] Carlos J Orihuela, Ulrich A Maus y Jeremy S Brown. *Can animal models really teach us anything about pneumonia?* Pro. 2020.
- [24] Licet Mena Valdés et al. «Efectos del D-005, extracto lipídico del fruto de Acrococmia crispa, sobre el daño pulmonar agudo inducido por ácido oleico en ratones». En: *Revista CENIC. Ciencias Biológicas* 50.1 (2019), págs. 62-80.
- [25] Shengle Qin et al. «Review of selected animal models for respiratory coronavirus infection and its application in drug research». En: *Journal of Medical Virology* 94.7 (2022), págs. 3032-3042.
- [26] Charles G Irvin y Jason HT Bates. «Measuring the lung function in the mouse: the challenge of size». En: *Respiratory research* 4 (2003).
- [27] Roberto Barrios. «Animal models of lung disease». En: *Basic Concepts of Molecular Pathology* (2009), págs. 153-157.

- [28] Rute FM Gomes y Jason HT Bates. «Geometric determinants of airway resistance in two isomorphic rodent species». En: *Respiratory physiology & neurobiology* 130.3 (2002), págs. 317-325.
- [29] Kent E Pinkerton et al. «Architecture of the tracheobronchial tree». En: (2015), págs. 33-51.
- [30] FM Bennett y SM Tenney. «Comparative mechanics of mammalian respiratory system». En: *Respiration physiology* 49.2 (1982), págs. 131-140.
- [31] Grzegorz Cieslewicz et al. «The late, but not early, asthmatic response is dependent on IL-5 and correlates with eosinophil infiltration». En: *The Journal of clinical investigation* 104.3 (1999), págs. 301-308.
- [32] Shinichiro Tomioka, Jason HT Bates y Charles G Irvin. «Airway and tissue mechanics in a murine model of asthma: alveolar capsule vs. forced oscillations». En: *Journal of Applied Physiology* 93.1 (2002), págs. 263-270.
- [33] Clarke G Tankersley, Richard Rabold y Wayne Mitzner. «Differential lung mechanics are genetically determined in inbred murine strains». En: *Journal of applied physiology* 86.6 (1999), págs. 1764-1769.
- [34] Scott Wagers et al. «Nonlinearity of respiratory mechanics during bronchoconstriction in mice with airway inflammation». En: *Journal of Applied Physiology* 92.5 (2002), págs. 1802-1807.
- [35] J Takezawa, FRED J Miller y JOHN J O'Neil. «Single-breath diffusing capacity and lung volumes in small laboratory mammals». En: *Journal of Applied Physiology* 48.6 (1980), págs. 1052-1059.
- [36] NICHOLAS J Gross. «Mechanical properties of mouse lungs: effects of degassing on normal, hyperoxic, and irradiated lungs». En: *Journal of Applied Physiology* 51.2 (1981), págs. 391-398.
- [37] W Mitzner, R Brown y W Lee. «In vivo measurement of lung volumes in mice». En: *Physiological genomics* 4.3 (2001), págs. 215-221.
- [38] Jeffrey S Reynolds, Victor J Johnson y David G Frazer. «Unrestrained acoustic plethysmograph for measuring specific airway resistance in mice». En: *Journal of Applied Physiology* 105.2 (2008), págs. 711-717.
- [39] Muriel Pichavant et al. «Animal models of airway sensitization». En: *Current protocols in immunology* 79.1 (2007), págs. 15-18.
- [40] Heinz Gerd Hoymann. «Invasive and noninvasive lung function measurements in rodents». En: *Journal of pharmacological and toxicological methods* 55.1 (2007), págs. 16-26.
- [41] Lennart KA Lundblad et al. «A reevaluation of the validity of unrestrained plethysmography in mice». En: *Journal of applied physiology* 93.4 (2002), págs. 1198-1207.

- [42] E Hamelmann et al. «Noninvasive measurement of airway responsiveness in allergic mice using barometric plethysmography». En: *American journal of respiratory and critical care medicine* 156.3 (1997), págs. 766-775.
- [43] Ferenc Petak et al. «Hyperoxia-induced changes in mouse lung mechanics: forced oscillations vs. barometric plethysmography». En: *Journal of Applied Physiology* 90.6 (2001), págs. 2221-2230.
- [44] CG Tankersley et al. «Hypercapnic ventilatory responses in mice differentially susceptible to acute ozone exposure». En: *Journal of Applied Physiology* 75.6 (1993), págs. 2613-2619.
- [45] Kurt H Albertine et al. «Temporal correlation of measurements of airway hyperresponsiveness in ovalbumin-sensitized mice». En: *American Journal of Physiology-Lung Cellular and Molecular Physiology* 283.1 (2002), págs. L219-L233.
- [46] Steven Weber. *The success of open source*. Harvard University Press, 2004.
- [47] Bruce Perens et al. «The open source definition». En: *Open sources: voices from the open source revolution* 1 (1999), págs. 171-188.
- [48] Richard M. Stallman. «Why “Free Software” is Better than “Open Source”». En: *Free Software, Free Society: Selected Essays of Richard M. Stallman* (2002). Ed. por Joshua Gay, págs. 57-62.
- [49] Rishab A Ghosh et al. *Free/libre and open source software: Survey and study*. 2002.
- [50] Karim R Lakhani y Eric Von Hippel. «How open source software works:“free” user-to-user assistance». En: *Produktentwicklung mit virtuellen Communities*. Springer, 2004, págs. 303-339.
- [51] Eric von Hippel y Georg von Krogh. «Open source software and the “private-collective” innovation model: Issues for organization science». En: *Organization science* 14.2 (2003), págs. 209-223.
- [52] Jérémie Bonvoisin et al. «What is the “Source” of Open Source Hardware?» En: *Journal of Open Hardware* 1 (1 sep. de 2017).
- [53] Open Source Hardware Association. *Definition of Open Source Hardware*. 2019. url: <https://www.oshwa.org/definition/>.
- [54] Bas Wijnen et al. «Open-source syringe pump library». En: *PloS one* 9.9 (2014), e107216.
- [55] Pankaj N Shrirao, Rajeshkumar U Sambhe y Pradip R Bodade. «Convective heat transfer analysis in a circular tube with different types of internal threads of constant pitch». En: *International Journal of Engineering and Advanced Technology* 2.3 (2013), págs. 335-340.
- [56] Jonathan E Elliott et al. «Aging-related changes in respiratory system mechanics and morphometry in mice». En: *American Journal of Physiology-Lung Cellular and Molecular Physiology* 311.1 (2016), págs. L167-L176.

- [57] Ariadna S Soto et al. «Toxoplasma gondii serine-protease inhibitor-1: A new adjuvant candidate for asthma therapy». En: *Plos one* 12.10 (2017), e0187002.
- [58] Carlos D Zappia et al. «Azelastine potentiates antiasthmatic dexamethasone effect on a murine asthma model». En: *Pharmacology research & perspectives* 7.6 (2019), e00531.
- [59] Ariadna Soto et al. «Contribution of Kazal-Like Domains of the Serine Protease Inhibitor-1 from Toxoplasma gondii in Asthma Therapeutic Vaccination Effectiveness». En: *International Archives of Allergy and Immunology* 183.5 (2022).
- [60] Marcelo Vivolo Aun et al. «Animal models of asthma: utility and limitations». En: *Journal of asthma and allergy* (2017), págs. 293-301.
- [61] Stephen T Holgate et al. «Asthma (Primer)». En: *Nature Reviews: Disease Primers* 1.1 (2015).
- [62] CL Gaultier et al. «Lung mechanics in rachitic rats». En: *American Review of Respiratory Disease* 130.6 (1984), págs. 1108-1110.
- [63] Rama Satyanarayana Raju Kalidhindi et al. «Role of estrogen receptors α and β in a murine model of asthma: exacerbated airway hyperresponsiveness and remodeling in ER β knockout mice». En: *Frontiers in Pharmacology* 10 (2020), pág. 1499.
- [64] Fien C Devos et al. «Forced expiration measurements in mouse models of obstructive and restrictive lung diseases». En: *Respiratory research* 18.1 (2017), págs. 1-14.
- [65] Joanne L Wright, Manuel Cosio y Andrew Churg. «Animal models of chronic obstructive pulmonary disease». En: *american journal of physiology-lung cellular and molecular physiology* 295.1 (2008), págs. L1-L15.
- [66] Alicia M Braxton et al. «Hamsters as a model of severe acute respiratory syndrome coronavirus-2». En: *Comparative medicine* 71.5 (2021), págs. 398-410.
- [67] Christoph Curths, Sascha Knauf y Franz-Josef Kaup. «Respiratory animal models in the common marmoset (*Callithrix jacchus*)». En: *Veterinary Sciences* 1.1 (2014), págs. 63-76.
- [68] Thomas Glaab et al. «Invasive and noninvasive methods for studying pulmonary function in mice». En: *Respiratory research* 8.1 (2007), págs. 1-10.
- [69] Jason HT Bates y Charles G Irvin. «Measuring lung function in mice: the phenotyping uncertainty principle». En: *Journal of applied physiology* 94.4 (2003).
- [70] Wayne Mitzner y Clarke Tankersley. «Interpreting Penh in mice». En: *Journal of applied physiology* 94.2 (2003), págs. 828-832.

Anexos

Anexo programas

Programa de Experimentación

Este programa se encuentra publicado en un repositorio libre de git-hub.
(<https://github.com/ifenoy/Lung-function>).

Para la implementación del programa de experimentación se utilizaron diferentes bibliotecas de Python. En particular, se utilizó la biblioteca *threading* para generar un hilo independiente que se encarga de leer el puerto UART y cargar los datos recibidos en una base de datos. De esta manera, se logra una separación entre la adquisición de datos y el procesamiento de los mismos, lo que permite una mayor eficiencia y organización en el programa.

Además, se utilizó la biblioteca *tkinter* para desarrollar la interfaz gráfica intuitiva y fácil de usar para el usuario. A través de esta interfaz, el usuario puede configurar los parámetros de ventilación y visualizar en tiempo real los valores recibidos del Arduino.

Para la visualización de los datos en tiempo real, se utilizó la función de *animation* de la biblioteca *matplotlib*. De esta forma, se logró una visualización animada y dinámica de los valores de los sensores, lo que permite una mejor comprensión y seguimiento del proceso de ventilación.

Finalmente, con la biblioteca *pandas* se almacenaron los datos adquiridos en un archivo CSV, lo que permite su posterior procesamiento y análisis de forma sencilla y eficiente. En resumen, la combinación de estas bibliotecas permitió el desarrollo de un programa completo y eficiente para la adquisición y visualización de datos en tiempo real durante la experimentación, así como su posterior almacenamiento y análisis.

```

import numpy as np

import matplotlib.pyplot as plt
import matplotlib.animation as animation

import csv

from matplotlib.backends.backend_tkagg import (
    FigureCanvasTkAgg, NavigationToolbar2Tk)

import tkinter as tk
from tkinter import ttk
from tkinter.messagebox import showerror
from threading import Thread

import pandas as pd


class GetData(Thread):
    def __init__(self, out_data, puerto, SegundosMostrados, TiempoMuestreo):
        super().__init__()
        try:
            self.serialArduino = serial.Serial(puerto, 115200, timeout=1.0)
        except:
            showerror(title='Error',
                      message='Error en la conexión con el arduino')
            self.Muestrand = False
            return
        self.out_data = out_data
        self.SegundosMostrados = SegundosMostrados
        self.TiempoMuestreo = TiempoMuestreo
        self.Nmuestas = int(SegundosMostrados / TiempoMuestreo)
        self.Muestrand = True
        self.line = ()

    def run(self):
        while self.Muestrand:
            self.column = 0
            try:
                self.line = self.serialArduino.readline().decode('ascii')

                for i in self.line.split(","): # Recorremos los campos que hemos recibido
                    if self.column < 4:
                        self.out_data[self.column].append(float(i)) # Asignamos a cada columna de los datos para pintar, la columna correspondiente del input
                        self.column = self.column + 1

                self.out_data[4].append(float(self.out_data[2][self.Nmuestas]) - float(self.out_data[1][self.Nmuestas]))
            except:
                pass

            if len(self.out_data[0]) > self.Nmuestas:
                self.out_data[5].append((self.out_data[5][-1]) + (self.out_data[0][-1]) * self.TiempoMuestreo - (self.out_data[0][0]) * self.TiempoMuestreo)
                self.out_data[0].pop(0)
            if len(self.out_data[1]) > self.Nmuestas:
                self.out_data[1].pop(0)
            if len(self.out_data[2]) > self.Nmuestas:
                self.out_data[2].pop(0)
            if len(self.out_data[3]) > self.Nmuestas:
                self.out_data[3].pop(0)
            if len(self.out_data[4]) > self.Nmuestas:
                self.out_data[4].pop(0)
            if len(self.out_data[5]) > self.Nmuestas:
                self.out_data[5].pop(0)

    def _desconectar(self):
        self.Muestrand = False
        self.serialArduino.flushInput()
        self.serialArduino.flushOutput()
        self.serialArduino.close()

    def _enviar(self, cad):
        self.serialArduino.write(cad.encode('ascii'))

```

```

class Entorno(tk.Tk):
    def __init__(self):
        super().__init__()

        self.title('Programa')

        self._leer_set()

        self.SegundosMostrados=float(self.Set['SegundosMostrados'])

        self.VolumenArduino=float(self.Set['VolumenArduino'])
        self.VelocidadInicial=float(self.Set['VelocidadInicial'])

        self.PresionMaxima=float(self.Set['PresionMaxima'])

        self.TiempoMuestreo=float(self.Set['TiempoMuestreo'])

        self.Nmuestas=int(self.SegundosMostrados/self.TiempoMuestreo)

        self.gData = [[0] * self.Nmuestas, [0] * self.Nmuestas]

        self.Puerto = self.Set['Puerto']

        self.create_header_frame()
        self.create_body_plot()
        self.create_body_frame()

    def create_header_frame(self):

        self.header = ttk.Frame(self)
        # configure the grid
        self.header.columnconfigure(0, weight=1)
        self.header.columnconfigure(1, weight=1)
        self.header.columnconfigure(2, weight=1)
        self.header.columnconfigure(3, weight=1)

        # Conectar button
        self.conectar_button = ttk.Button(self.header, text='Conectar')
        self.conectar_button['command'] = self._conectar
        self.conectar_button.grid(column=0, row=1)
        # Desconectar button
        self.desconectar_button = ttk.Button(self.header, text='Desconectar')
        self.desconectar_button['command'] = self._desconectar
        self.desconectar_button.grid(column=1, row=1)
        # Salir button
        self.salir_button = ttk.Button(self.header, text='Salir')
        self.salir_button['command'] = self._quit
        self.salir_button.grid(column=2, row=1)

        # attach the header frame
        self.header.grid(column=0, row=0, sticky=tk.NSEW)

    def create_body_plot(self):

        self.fig_flujo_presion_steps_volumen, (self.ax_flujo, self.ax_presion, self.ax_steps) = plt.subplots(3,1,figsize=(5,10))
        self.fig_flujo_presion_steps_volumen.subplots_adjust(left = 0.2,hspace = 0.4)
        self.fig_resistencia_compliance, (self.ax_resistencia, self.ax_compliance) = plt.subplots(2,1,figsize=(5,10))
        self.fig_resistencia_compliance, (self.ax_compliance) = plt.subplots(figsize=(5,10))
        self.fig_resistencia_compliance.subplots_adjust(left = 0.2,hspace = 0.4)
        plt.rcdefaults()

        # initialize two line objects (one in each axes)
        self.line_flujo, = self.ax_flujo.plot([], [], lw=2)
        self.line_presion, = self.ax_presion.plot([], [], lw=2, color='r')
        self.line_steps, = self.ax_steps.plot([], [], lw=2, color='g')
        self.hl_flujo_presion_steps_volumen = [self.line_flujo, self.line_presion, self.line_steps]

```

```

self.line_resistencia, = self.ax_resistencia.plot([], [], lw=2)
self.line_compliance, = self.ax_compliance.plot([], [], lw=2, color='r')
self.hl_resistencia_compliance = [self.line_resistencia, self.line_compliance]
self.hl_resistencia_compliance = [self.line_compliance]

self.ax_flujo.set_ylim(-10, 10)
self.ax_flujo.set_xlim(0, self.SegundosMostrados)
self.ax_flujo.grid()
self.ax_flujo.set_xlabel('t[s]')
self.ax_flujo.set_ylabel('Flujo[ml/s]')
self.ax_flujo.set_title('Flujo sensado')

self.ax_presion.set_ylim(-5, 7)
self.ax_presion.set_xlim(0, self.SegundosMostrados)
self.ax_presion.grid()
self.ax_presion.set_xlabel('t[s]')
self.ax_presion.set_ylabel('Presion[kPa]')
self.ax_presion.set_title('Presion sensada')

self.ax_steps.set_ylim(0, 5)
self.ax_steps.set_xlim(0, self.SegundosMostrados)
self.ax_steps.grid()
self.ax_steps.set_xlabel('t[s]')
self.ax_steps.set_ylabel('Volumen[ml]')
self.ax_steps.set_title('Volumen ventilado')

self.ax_resistencia.set_ylim(-5, 5)
self.ax_resistencia.set_xlim(-2, 5)
self.ax_resistencia.grid()
self.ax_resistencia.set_xlabel('Presion[kPa]')
self.ax_resistencia.set_ylabel('Flujo[ml/s]')
self.ax_resistencia.set_title('Flujo vs Presion sensada')

self.ax_compliance.set_ylim(-1, 5)
self.ax_compliance.set_xlim(-5, 7)
self.ax_compliance.grid()
self.ax_compliance.set_xlabel('Presion[kPa]')
self.ax_compliance.set_ylabel('Volumen[ml]')
self.ax_compliance.set_title('Volumen ventilado vs Presion sensada')

self.canvas_flujo_presion_steps_volumen = FigureCanvasTkAgg(self.fig_flujo_presion_steps_volumen, master=self)
self.canvas_flujo_presion_steps_volumen.draw()

self.canvas_resistencia_compliance = FigureCanvasTkAgg(self.fig_resistencia_compliance, master=self)
self.canvas_resistencia_compliance.draw()

self.toolbar_flujo_presion_volumen = NavigationToolbar2Tk(self.canvas_flujo_presion_steps_volumen, self, pack_toolbar=False)
self.toolbar_flujo_presion_volumen.update()

self.toolbar_resistencia_compliance = NavigationToolbar2Tk(self.canvas_resistencia_compliance, self, pack_toolbar=False)
self.toolbar_resistencia_compliance.update()

self.canvas_flujo_presion_steps_volumen.get_tk_widget().grid(column=0, row=1)
self.canvas_resistencia_compliance.get_tk_widget().grid(column=1, row=1)

self.ani_flujo_presion_steps_volumen = animation.FuncAnimation(self.fig_flujo_presion_steps_volumen, update_line_flujo_presion_steps_volumen, fargs=(self.hl_resistencia_compliance))
self.ani_resistencia_compliance = animation.FuncAnimation(self.fig_resistencia_compliance, update_line_resistencia_compliance, fargs=(self.hl_resistencia_compliance))

def create_body_frame(self):
    self.body = ttk.Frame(self)

    self.header.rowconfigure(0, weight=2)
    self.header.rowconfigure(1, weight=2)
    self.header.rowconfigure(2, weight=2)
    self.header.rowconfigure(3, weight=2)

    # salvar button

```

```

        self.salvar_button = ttk.Button(self.body, text='Salvar')
        self.salvar_button['command'] = self._salvar
        self.salvar_button.grid(column=3, row=0)

    # Set Arduino button
    self.setArduino_button = ttk.Button(self.body, text='Set@Arduino')
    self.setArduino_button['command'] = self._set_arduino
    self.setArduino_button.grid(column=4, row=0)

    # Set Stop Arduino button
    self.stopArduino_button = ttk.Button(self.body, text='Stop@Arduino')
    self.stopArduino_button['command'] = self._stop_arduino
    self.stopArduino_button.grid(column=5, row=0)

    # set button
    self.set_button = ttk.Button(self.body, text='set')
    self.set_button['command'] = self._set
    self.set_button.grid(column=2, row=0)

    #texto tiempo
    self.label = ttk.Label(self.body, text='segundos')
    self.label.grid(column=2, row=2)

    #entry tiempo
    self.tiempo_var = tk.IntVar(value=self.Set['SegundosMostrados'])
    self.tiempo_entry = ttk.Entry(self.body,
                                  textvariable=self.tiempo_var,
                                  width=10)

    self.tiempo_entry.grid(column=2, row=3)

    #texto Volumen arduino
    self.label = ttk.Label(self.body, text='Volumen@Arduino')
    self.label.grid(column=3, row=2)

    #entry Volumens arduino
    self.VolumenArduino_var = tk.StringVar(value=self.Set['VolumenArduino'])
    self.VolumenArduino_entry = ttk.Entry(self.body,
                                         textvariable=self.VolumenArduino_var,
                                         width=10)

    self.VolumenArduino_entry.grid(column=3, row=3)

    #texto VelocidadInicial
    self.label = ttk.Label(self.body, text='Velocidad@Inicial')
    self.label.grid(column=4, row=2)

    #entry VelocidadInicial
    self.VelocidadInicial_var = tk.StringVar(value=self.Set['VelocidadInicial'])
    self.VelocidadInicial_entry = ttk.Entry(self.body,
                                         textvariable=self.VelocidadInicial_var,
                                         width=10)

    self.VelocidadInicial_entry.grid(column=4, row=3)

    #texto VelocidadInicial
    self.label = ttk.Label(self.body, text='Presion@Max')
    self.label.grid(column=5, row=2)

    #entry VelocidadInicial
    self.PresionMaxima_var = tk.StringVar(value=self.Set['PresionMaxima'])
    self.PresionMaxima_entry = ttk.Entry(self.body,
                                         textvariable=self.PresionMaxima_var,
                                         width=10)

    self.PresionMaxima_entry.grid(column=5, row=3)

    # attach the body frame
    self.body.grid(column=1, row=0)

def _conectarar(self):
    # Configuramos y lanzamos el hilo encargado de leer datos del serial
    try:
        if self.Muestreo.is_alive():
            pass
        else:
            self.Muestreo = GetData(self.gData, self.Puerto, self.SegundosMostrados, self.TiempoMuestreo)
    
```

```

        self.Muestreo.start()
    except:
        try:
            self.Muestreo = GetData(self.gData, self.Puerto, self.SegundosMostrados, self.TiempoMuestreo)
            self.Muestreo.start()
        except:
            pass

    def _desconectar(self):
        try:
            if self.Muestreo.is_alive():
                self.Muestreo._desconectar()
        except:
            pass

    def _salvar(self):
        self.path= tk.filedialog.asksaveasfilename(title = "Select A File", filetypes = (( "All Files", "*.*"), ("Python files", "*.png"), ("mp4", "*mp4"), ("wmv", ".wmv")))
        os.mkdir(self.path)
        with open(self.path+'/Data.txt', 'w') as self.file:
            writer = csv.writer(self.file,delimiter=',')
            writer.writerow(['SegundosMostrados', str(self.Set['SegundosMostrados'])])
            writer.writerow(['TiempoMuestreo', str(self.Set['TiempoMuestreo'])])
            writer.writerow(['Puerto', str(self.Set['Puerto'])])
            writer.writerow(['VolumenArduino', str(self.Set['VolumenArduino'])])
            writer.writerow(['VelocidadInicial', str(self.Set['VelocidadInicial'])])
            writer.writerow(['PresionMaxima', str(self.Set['PresionMaxima'])])
        self.file.close()

        with open(self.path+'/Set.txt', 'w') as self.file:
            writer = csv.writer(self.file,delimiter=',')
            writer.writerow(['SegundosMostrados', str(self.Set['SegundosMostrados'])])
            writer.writerow(['TiempoMuestreo', str(self.Set['TiempoMuestreo'])])
            writer.writerow(['Puerto', str(self.Set['Puerto'])])
            writer.writerow(['VolumenArduino', str(self.Set['VolumenArduino'])])
            writer.writerow(['VelocidadInicial', str(self.Set['VelocidadInicial'])])
            writer.writerow(['PresionMaxima', str(self.Set['PresionMaxima'])])
        self.file.close()

    def _set_arduino(self):

        if float(self.VolumenArduino_var.get())>0:
            self.Set['VolumenArduino']=float(self.VolumenArduino_var.get())
        if float(self.VelocidadInicial_var.get())>0:
            self.Set['VelocidadInicial']=float(self.VelocidadInicial_var.get())
        if float(self.PresionMaxima_var.get())>0:
            self.Set['PresionMaxima']=float(self.PresionMaxima_var.get())

        self.TiempoPaso= 1.0/(146*float(self.Set['VelocidadInicial'])) # ml por paso *velocidad = frecuencia // periodo=1/f

        print("Set," + str(self.Set['VolumenArduino'])+ ',' + str((self.TiempoPaso*1000000.0))+ ',' + str(self.Set['PresionMaxima']))
        try:
            if self.Muestreo.is_alive():
                self.Muestreo._enviar("Set," + str(self.Set['VolumenArduino'])+ ',' + str((self.TiempoPaso*1000000.0))+ ',' + str(self.Set['PresionMaxima']))
        except:
            showerror(title='Error',message="Arduino no conectado")

    def _stop_arduino(self):

        if self.Muestreo.is_alive():
            print("Set," + str(0)+ ',' + str(1000))
            self.Muestreo._enviar("Set," + str(0)+ ',' + str(1000))

    def _set(self):
        self._desconectar()

        try:
            if self.tiempo_var.get()>0:
                self.Set['SegundosMostrados']=self.tiempo_var.get()
        except:
            pass

        try:
            if float(self.VolumenArduino_var.get())>0:
                self.Set['VolumenArduino']=float(self.VolumenArduino_var.get())
            if float(self.VelocidadInicial_var.get())>0:
                self.Set['VelocidadInicial']=float(self.VelocidadInicial_var.get())
            if float(self.PresionMaxima_var.get())>0:
                self.Set['PresionMaxima']=float(self.PresionMaxima_var.get())
        except:

```

```

    pass

self._guardar_set()
self._restart()

def _leer_set(self):
    try:
        with open('setting/setting.csv') as self._file:
            self.rows = csv.reader(self._file)
            self.Set = {}
            for self.row in self.rows:
                if not self.row: # Saltea filas sin datos
                    continue
                self.row = [func(val) for func, val in zip([str,str], self.row) ]
                self.Set[self.row[0]] = self.row[1]
            self._file.close()

    except:
        showerror(title='Error', message="Error al leer el Set")

def _guardar_set(self):
    try:
        with open('setting/setting.csv', 'w') as self._file:
            writer = csv.writer(self._file, delimiter=',')
            writer.writerow(['SegundosMostrados', str(self.Set['SegundosMostrados'])])
            writer.writerow(['TiempoMuestreo', str(self.Set['TiempoMuestreo'])])
            writer.writerow(['Puerto', str(self.Set['Puerto'])])
            writer.writerow(['VolumenArduino', str(self.Set['VolumenArduino'])])
            writer.writerow(['VelocidadInicial', str(self.Set['VelocidadInicial'])])
            writer.writerow(['PresionMaxima', str(self.Set['PresionMaxima'])])
        self._file.close()

    except:
        showerror(title='Error', message="Error al guardar el Set")

def _restart(self):
    try:
        if self.Muestreo.is_alive():
            self.Muestreo._desconectar()
    except:
        pass

    self.quit() # stops mainloop
    self.destroy() # this is necessary on Windows to prevent
    sys.stdout.flush()
    os.system("experimentation_software.pyw1")

def _quit(self):
    try:
        if self.Muestreo.is_alive():
            self.Muestreo._desconectar()
    except:
        pass
    self.quit() # stops mainloop
    self.destroy() # this is necessary on Windows to prevent
    # Fatal Python Error: PyEval_RestoreThread: NULL tstate

def update_line_flujo_presion_steps_volumen(aux, hl_flujo_presion_steps_volumen, data, SegundosMostrados, TiempoMuestreo):
    hl_flujo_presion_steps_volumen[0].set_data(np.arange(0,SegundosMostrados,TiempoMuestreo), data[0])
    hl_flujo_presion_steps_volumen[1].set_data(np.arange(0,SegundosMostrados,TiempoMuestreo), data[4])
    hl_flujo_presion_steps_volumen[2].set_data(np.arange(0,SegundosMostrados,TiempoMuestreo), data[3])
    return hl_flujo_presion_steps_volumen

def update_line_resistencia_compliance(aux, hl_resistencia_compliance, data):
    hl_resistencia_compliance[0].set_data(data[4], data[3])
    return hl_resistencia_compliance,

```

```
App = Entorno()
App.mainloop()
```

Programa de Procesamiento

Este programa se encuentra publicado en un repositorio libre de git-hub.
(<https://github.com/ifenoy/Lung-function>).

Para lograr esto, se ha empleado las bibliotecas *numpy* y *scipy* que permiten realizar operaciones matemáticas avanzadas y análisis estadísticos sobre la base de datos de las mediciones almacenadas en el archivo CSV.

Todo este procesamiento se visualiza a través de gráficos generados con la biblioteca *matplotlib*.

Finalmente, con la biblioteca *pandas* se almacenaron los datos adquiridos en un archivo CSV, lo que permite su posterior procesamiento y análisis de forma sencilla y eficiente. En resumen, la combinación de estas bibliotecas permitió el desarrollo de un programa completo y eficiente para la adquisición y visualización de datos en tiempo real durante la experimentación, así como su posterior almacenamiento y análisis.

```

import os
import numpy as np
import matplotlib.pyplot as plt
import csv

from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
NavigationToolbar2Tk)

import tkinter as tk
from tkinter import ttk
from tkinter.messagebox import showerror

class Entorno(tk.Tk):
    def __init__(self):
        super().__init__()

        self.create_header_frame()

    def create_header_frame(self):
        self.header = ttk.Frame(self)

        # configure the grid
        self.header.columnconfigure(0, weight=1)
        self.header.columnconfigure(1, weight=1)
        self.header.columnconfigure(2, weight=1)
        self.header.columnconfigure(3, weight=1)

        # Leer button
        self.leer_button = ttk.Button(self.header, text='Leer')
        self.leer_button['command'] = self._leer
        self.leer_button.grid(column=1, row=1)

        # Salir button
        self.salir_button = ttk.Button(self.header, text='Salir')
        self.salir_button['command'] = self._quit
        self.salir_button.grid(column=2, row=1)

        # Entrada Label
        self.Lebel_var = tk.StringVar(value='Label')
        self.Lebel_entry = ttk.Entry(self.header,
                                     textvariable=self.Lebel_var,
                                     width=10)

        self.Lebel_entry.grid(column=3, row=1)

        self.Color_Combobox = ttk.Combobox(self.header,
                                           values=['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']
                                           )
        self.Color_Combobox.grid(column=4, row=1)

        # attach the header frame
        self.header.grid(column=0, row=0, sticky=tk.NSEW)

    def create_body_plot(self):
        # Agregar button
        self.agregar_button = ttk.Button(self.header, text='Aregar')
        self.agregar_button['command'] = self._agregar
        self.agregar_button.grid(column=5, row=1)

        self.form_titulos = {'fontname': 'calibri'}

        self.flujo_tiempo, self.ax_flujo_tiempo = plt.subplots(figsize=(5,4))

```

```

self.flujo_tiempo.subplots_adjust(left = 0.2, hspace = 0.1)

self.presion_tiempo, self.ax_presion_tiempo = plt.subplots(figsize=(5,4))
self.presion_tiempo.subplots_adjust(left = 0.2, hspace = 0.1)
self.volumen_ventilado_tiempo, self.ax_volumen_ventilado_tiempo = plt.subplots(figsize=(5,4))
self.volumen_ventilado_tiempo.subplots_adjust(left = 0.2, hspace = 0.1)
self.volumen_integrado_y_presion_tiempo, self.ax_volumen_integrado_y_presion_tiempo = plt.subplots(figsize=(5,4))
self.volumen_integrado_y_presion_tiempo.subplots_adjust(left = 0.2, hspace = 0.1)

self.presion_volumen, self.ax_presion_volumen = plt.subplots(figsize=(7,9))
self.presion_volumen.subplots_adjust(left = 0.2, hspace = 0.1)

self.flujo= self.gData[0]

self.t = np.arange(0,len(self.flujo)*float(self.Set['TiempoMuestreo']),float(self.Set['TiempoMuestreo']))

self.presion = self.gData[4]
self.volumen_ventilado=self.gData[3]
self.index=[]

self.auxJ =0;
self.auxK =0;
self.index_primer_0 =0;
while self.auxJ < len(self.volumen_ventilado):
    if self.volumen_ventilado[self.auxJ] == 0:
        self.index_primer_0 = self.auxJ;
        self.auxK = self.auxJ;
        while (self.auxK < len(self.volumen_ventilado)) and (self.volumen_ventilado[self.auxK] == 0) :
            self.auxK = self.auxK +1;
            self.index.append(int(self.auxJ+((self.auxK-self.auxJ)/2)))
        self.auxJ=self.auxK
        self.auxJ = self.auxJ +1;

self.ciclos_flujo =[]
self.ciclos_t=[]
self.ciclos_presion=[]
self.ciclos_volumen_ventilado=[]
self.n_ciclo = 0

for i in range(0,len(self.index)-1,2):
    self.ciclos_flujo.append(self.flujo[self.index[i]:self.index[i+1]])
    self.ciclos_t.append(np.arange(0,len(self.ciclos_flujo[self.n_ciclo])*float(self.Set['TiempoMuestreo']),float(self.Set['TiempoMuestreo']))
    self.ciclos_presion.append(self.presion[self.index[i]:self.index[i+1]])
    self.ciclos_volumen_ventilado.append(self.volumen_ventilado[self.index[i]:self.index[i+1]])
    self.n_ciclo = self.n_ciclo+1

self.ciclos_volumen_integrado=[]
for i in range(0,self.n_ciclo):
    self.aux = [element * float(self.Set['TiempoMuestreo']) for element in self.ciclos_flujo[i]]
    self.ciclos_volumen_integrado.append(np.cumsum(self.aux))

self.ax_flujo_tiempo.plot(self.t, self.flujo)

self.ax_flujo_tiempo.set_xlabel('t[s]')
self.ax_flujo_tiempo.set_ylabel('Flujo[m/s]')
self.ax_flujo_tiempo.set_title('Flujo\sensado', **self.form_titulos)
self.ax_flujo_tiempo.grid()

self.ax_presion_tiempo.plot(self.t, self.presion)

self.ax_presion_tiempo.set_xlabel('t[s]')
self.ax_presion_tiempo.set_ylabel('Presión[kPa]')
self.ax_presion_tiempo.set_title('Presión\sensada', **self.form_titulos)
self.ax_presion_tiempo.grid()

```

```

self.ax_volumen_ventilado_tiempo.plot(self.t, self.volumen_ventilado)
self.ax_volumen_ventilado_tiempo.set_xlabel('t[s]')
self.ax_volumen_ventilado_tiempo.set_ylabel('Volumen[ml]')
self.ax_volumen_ventilado_tiempo.set_title('VolumenVentilado', **self.form_titulos)
self.ax_volumen_ventilado_tiempo.grid()

self.ax_volumen_integrado_y_presion_tiempo.plot(self.ciclos_volumen_integrado[0], self.Color_Combo.get(), label=self.Lebel_var.get())
for i in range(1, self.n_ciclo):
    self.ax_volumen_integrado_y_presion_tiempo.plot(self.ciclos_volumen_integrado[i], self.Color_Combo.get())

self.ax_volumen_integrado_y_presion_tiempo.set_xlabel('t[s]')
self.ax_volumen_integrado_y_presion_tiempo.set_ylabel('Volumen[ml]')
self.ax_volumen_integrado_y_presion_tiempo.set_title('VolumenIntegrado', **self.form_titulos)
self.ax_volumen_integrado_y_presion_tiempo.grid()
self.ax_volumen_integrado_y_presion_tiempo.legend()

self.pendientes = []
self.ordenada_origen = []
self.max_presion = []
self.min_presion = []
self.max_volumen = []
self.index_max_presion = []
self.index_min_presion = []
self.index_max_volumen = []
self.index_080_max_presion = []
self.index_025_presion = []
for i in range(self.n_ciclo):

    self.ciclos_presion_np = np.array(self.ciclos_presion[i])
    self.ciclos_volumen_np = np.array(self.ciclos_volumen_integrado[i])
    self.max_presion.append(np.amax(self.ciclos_presion_np))
    self.min_presion.append(np.nanmin(self.ciclos_presion_np))
    self.max_volumen.append(np.amax(self.ciclos_volumen_np))
    self.index_max_presion.append(self.ciclos_presion[i].index(self.max_presion[i]))
    self.index_min_presion.append(self.ciclos_presion[i].index(self.min_presion[i]))
    self.index_max_volumen.append(int((np.where(self.ciclos_volumen_integrado[i] == self.max_volumen[i])[0])[0]))

    self.index_080_max_presion.append((np.abs(self.ciclos_presion_np[self.index_max_presion[i]:] - 0.8).argmin() + self.index_max_presion[i]))
    self.index_025_presion.append((np.abs(self.ciclos_presion_np[self.index_max_presion[i]:] - 0.25).argmin() + self.index_max_presion[i]))

    self.pendientes.append((self.ciclos_volumen_integrado[i][self.index_025_presion[i]] - self.ciclos_volumen_integrado[i][self.index_080_max_presion[i]]) / (self.ordenada_origen.append(self.ciclos_volumen_integrado[i][self.index_025_presion[i]] - self.pendientes[i] * self.ciclos_presion[i][self.index_025_p

print('Las pendientes son:')
print(self.pendientes)
print('El promedio es')
self.promedio = 0
self.error_estandar = 0
self.desviacion_estandar = 0
self.error_estandar = 0
for element in self.pendientes:
    self.promedio = self.promedio + element
self.promedio = self.promedio / self.n_ciclo
print(self.promedio)

for element in self.pendientes:
    self.desviacion_estandar = self.desviacion_estandar + (element - self.promedio) * (element - self.promedio)
self.desviacion_estandar = np.sqrt(self.desviacion_estandar / self.n_ciclo)
self.error_estandar = float(self.desviacion_estandar) / float(np.sqrt(self.n_ciclo))

self.promedio = 0
self.error_estandar = 0
self.desviacion_estandar = 0
self.error_estandar = 0
for element in self.pendientes:
    self.promedio = self.promedio + element
self.promedio = self.promedio / self.n_ciclo
print(self.promedio)

for element in self.pendientes:
    self.desviacion_estandar = self.desviacion_estandar + (element - self.promedio) * (element - self.promedio)
self.desviacion_estandar = np.sqrt(self.desviacion_estandar / self.n_ciclo)

```

```

self.error_estandar = float(self.desviacion_estandar) / float(np.sqrt(self.n_ciclo))

aux_t = np.arange(-1,2,0.01)
i=0

self.ax_presion_volumen.plot(self.ciclos_presion[i],self.ciclos_volumen_integrado[i],self.Color_Combo.get(), label=self.Lebel_var.get())
for i in range(1,self.n_ciclo):
    self.ax_presion_volumen.plot(self.ciclos_presion[i],self.ciclos_volumen_integrado[i],self.Color_Combo.get())

self.ax_presion_volumen.plot(aux_t, aux_t* self.pendientes[0] + (self.ordenada_origen[0]),'k--')

self.ax_presion_volumen.set_ylabel('Volumen[m³]')
self.ax_presion_volumen.set_xlabel('Presión[kPa]')
self.ax_presion_volumen.set_title('Volumen integrado vs Presión sensada', **self.form_titulos)
self.ax_presion_volumen.grid()
self.ax_presion_volumen.legend()

self.canvas_flujo_tiempo = FigureCanvasTkAgg(self.flujo_tiempo, master=self) # A tk.DrawingArea.
self.canvas_flujo_tiempo.draw()
self.canvas_flujo_tiempo.get_tk_widget().grid(column=0, row=1)

self.canvas_presion_tiempo = FigureCanvasTkAgg(self.presion_tiempo, master=self) # A tk.DrawingArea.
self.canvas_presion_tiempo.draw()
self.canvas_presion_tiempo.get_tk_widget().grid(column=0, row=3)

self.canvas_volumen_integrado_y_presion_tiempo = FigureCanvasTkAgg(self.volumen_integrado_y_presion_tiempo, master=self) # A tk.DrawingArea
self.canvas_volumen_integrado_y_presion_tiempo.draw()
self.canvas_volumen_integrado_y_presion_tiempo.get_tk_widget().grid(column=1, row=1)

self.canvas_presion_volumen = FigureCanvasTkAgg(self.presion_volumen, master=self) # A tk.DrawingArea.
self.canvas_presion_volumen.draw()
self.canvas_presion_volumen.get_tk_widget().grid(column=2, row=1, rowspan=3)

self.toolbar_flujo_tiempo = ttk.Frame(self)
self.toolbar_canvas_flujo_tiempo = NavigationToolbar2Tk(self.canvas_flujo_tiempo, self.toolbar_flujo_tiempo )
self.toolbar_flujo_tiempo.grid(column=0, row=2, sticky=tk.NSEW)
self.toolbar_canvas_flujo_tiempo.update()

self.toolbar_presion_tiempo = ttk.Frame(self)
self.toolbar_canvas_presion_tiempo = NavigationToolbar2Tk(self.canvas_presion_tiempo, self.toolbar_presion_tiempo)
self.toolbar_presion_tiempo.grid(column=0, row=4, sticky=tk.NSEW)
self.toolbar_canvas_presion_tiempo.update()

self.toolbar_volumen_integrado_y_presion_tiempo = ttk.Frame(self)
self.toolbar_canvas_volumen_integrado_y_presion_tiempo = NavigationToolbar2Tk(self.canvas_volumen_integrado_y_presion_tiempo, self.toolbar_volumen_integrado_y_presion_tiempo)
self.toolbar_volumen_integrado_y_presion_tiempo.grid(column=1, row=2, sticky=tk.NSEW)
self.toolbar_canvas_volumen_integrado_y_presion_tiempo.update()

self.toolbar_presion_volumen = ttk.Frame(self)
self.toolbar_canvas_presion_volumen = NavigationToolbar2Tk(self.canvas_presion_volumen, self.toolbar_presion_volumen)
self.toolbar_presion_volumen.grid(column=2, row=4, sticky=tk.NSEW)
self.toolbar_canvas_presion_volumen.update()

self.body_text = ttk.Frame(self)

# text and scrollbar
self.texto = tk.Text(self.body_text, height=25,width=60)
self.texto.grid(column=0, row=1)

scrollbar = ttk.Scrollbar(self.body_text,
                           orient='vertical',
                           command=self.texto.yview)

scrollbar.grid(column=1, row=1, sticky=tk.NS)
self.texto['yscrollcommand'] = scrollbar.set

# attach the body frame
self.body_text.grid(column=1, row=3, sticky=tk.NSEW, padx=5, pady=5)

self.texto.insert(tk.END, 'Las pendientes de ' + self.Lebel_var.get() + ' son: ' + str(self.pendientes) + 'ml/kPa\n')

self.texto.insert(tk.END, 'El promedio es: ')
self.promedio_pendientes = 0
self.promedio_volumen = 0
self.error_estandar = 0
self.desviacion_estandar = 0
self.error_estandar = 0
for element in self.pendientes:
    self.promedio_pendientes = self.promedio_pendientes + element
self.promedio_pendientes = self.promedio_pendientes / self.n_ciclo

```

```

self.texto.insert(tk.END, str(self.promedio_pendientes) + 'ml/kPa\n')

self.texto.insert(tk.END, 'Las IC de ' + self.Lebel_var.get() + ' son: ' + str(self.max_volumen) + 'ml\n')

for element in self.max_volumen:
    self.promedio_volumen = self.promedio_volumen + element
self.promedio_volumen = self.promedio_volumen / self.n_ciclo
self.texto.insert(tk.END, 'El promedio es: ')
self.texto.insert(tk.END, str(self.promedio_volumen) + 'ml\n')
self.texto.insert(tk.END, '-----\n')

def agregar_plot(self):
    self.flujo= self.gData[0]

    self.t = np.arange(0,len(self.flujo)*float(self.Set['TiempoMuestreo']),float(self.Set['TiempoMuestreo']))

    self.presion = self.gData[4]
    self.volumen_ventilado=self.gData[3]
    self.index=[]

    self.auxJ =0;
    self.auxK =0;
    self.index_primer_0 =0;
    while self.auxJ < len(self.volumen_ventilado):
        if self.volumen_ventilado[self.auxJ] == 0:
            self.index_primer_0 = self.auxJ;
            self.auxK = self.auxJ;
            while (self.auxK < len(self.volumen_ventilado)) and (self.volumen_ventilado[self.auxK] == 0) :
                self.auxK = self.auxK +1;
            self.index.append(int(self.auxJ+((self.auxK-self.auxJ)/2)))
            self.auxJ=self.auxK
        self.auxJ = self.auxJ +1;

    self.ciclos_flujo =[]
    self.ciclos_t=[]
    self.ciclos_presion =[]
    self.ciclos_volumen_ventilado =[]
    self.n_ciclo = 0

    for i in range(0,len(self.index)-1,2):
        self.ciclos_flujo.append(self.flujo[self.index[i]:self.index[i+1]])
        self.ciclos_t.append(np.arange(0,len(self.ciclos_flujo[self.n_ciclo])*float(self.Set['TiempoMuestreo']),float(self.Set['TiempoMuestreo'])))
        self.ciclos_presion.append(self.presion[self.index[i]:self.index[i+1]])
        self.ciclos_volumen_ventilado.append(self.volumen_ventilado[self.index[i]:self.index[i+1]])
        self.n_ciclo = self.n_ciclo+1

    print(len(self.ciclos_flujo))

    self.ciclos_volumen_integrado =[]
    for i in range(0,self.n_ciclo):
        self.aux = [element * float(self.Set['TiempoMuestreo']) for element in self.ciclos_flujo[i]]
        self.ciclos_volumen_integrado.append(np.cumsum(self.aux))

self.ax_flujo_tiempo.plot(self.t,self.flujo)

self.ax_flujo_tiempo.set_xlabel('t[s]')
self.ax_flujo_tiempo.set_ylabel('Flujo[ml/s]')
self.ax_flujo_tiempo.set_title('Flujo sensado', **self.form_titulos)

self.ax_presion_tiempo.plot(self.t,self.presion)

self.ax_presion_tiempo.set_xlabel('t[s]')
self.ax_presion_tiempo.set_ylabel('Presión[kPa]')
self.ax_presion_tiempo.set_title('Presión sensada', **self.form_titulos)

```

```

        self.ax_volumen_ventilado_tiempo.plot(self.t, self.volumen_ventilado)
        self.ax_volumen_ventilado_tiempo.set_xlabel('t[s]')
        self.ax_volumen_ventilado_tiempo.set_ylabel('Volumen[ml]')
        self.ax_volumen_ventilado_tiempo.set_title('VolumenVentilado', **self.form_titulos)

    for i in range(0, self.n_ciclo):
        self.ax_volumen_integrado_y_presion_tiempo.plot(self.ciclos_volumen_integrado[i])

        self.ax_volumen_integrado_y_presion_tiempo.set_xlabel('t[s]')
        self.ax_volumen_integrado_y_presion_tiempo.set_ylabel('Volumen[ml]')
        self.ax_volumen_integrado_y_presion_tiempo.set_title('VolumenIntegrado', **self.form_titulos)
        self.ax_volumen_integrado_y_presion_tiempo.grid()

    self.pendientes = []
    self.ordenada_origen = []
    self.max_presion = []
    self.min_presion = []
    self.max_volumen = []
    self.index_max_presion = []
    self.index_min_presion = []
    self.index_max_volumen = []
    self.index_080_max_presion = []
    self.index_025_presion = []

    for i in range(self.n_ciclo):

        self.ciclos_presion_np = np.array(self.ciclos_presion[i])
        self.ciclos_volumen_np = np.array(self.ciclos_volumen_integrado[i])
        self.max_presion.append(np.amax(self.ciclos_presion_np))
        self.min_presion.append(np.nanmin(self.ciclos_presion_np))
        self.max_volumen.append(np.amax(self.ciclos_volumen_np))

        self.index_max_presion.append(self.ciclos_presion[i].index(self.max_presion[i]))
        self.index_min_presion.append(self.ciclos_presion[i].index(self.min_presion[i]))

        self.index_max_volumen.append(int((np.where(self.ciclos_volumen_integrado[i] == self.max_volumen[i])[0])[0]))

        self.index_080_max_presion.append((np.abs(self.ciclos_presion_np[self.index_max_presion[i]:] - 0.8).argmin()) + self.index_max_presion[i])
        self.index_025_presion.append((np.abs(self.ciclos_presion_np[self.index_max_presion[i]:] - 0.25).argmin()) + self.index_max_presion[i])

        self.pendientes.append((self.ciclos_volumen_integrado[i][self.index_025_presion[i]] - self.ciclos_volumen_integrado[i][self.index_080_max_presion[i]]) / (self.index_080_max_presion[i] - self.index_025_presion[i]))
        self.ordenada_origen.append(self.ciclos_volumen_integrado[i][self.index_025_presion[i]] - self.pendientes[i] * self.ciclos_presion[i][self.index_025_presion[i]])

    print('Las pendientes son:')
    print(self.pendientes)
    print('El promedio es')
    self.promedio = 0
    self.error_estandar = 0
    self.desviacion_estandar = 0
    self.error_estandar = 0
    for element in self.pendientes:
        self.promedio = self.promedio + element
    self.promedio = self.promedio / self.n_ciclo
    print(self.promedio)

    for element in self.pendientes:
        self.desviacion_estandar = self.desviacion_estandar + (element - self.promedio) * (element - self.promedio)

    self.desviacion_estandar = np.sqrt(self.desviacion_estandar / self.n_ciclo)

    self.error_estandar = float(self.desviacion_estandar) / float(np.sqrt(self.n_ciclo))

    aux_t = np.arange(-1, 2, 0.01)
    i=0

    self.ax_presion_volumen.plot(self.ciclos_presion[i], self.ciclos_volumen_integrado[i], self.Color_Combo.get(), label=self.Lebel_var.get())

    for i in range(1, self.n_ciclo):
        self.ax_presion_volumen.plot(self.ciclos_presion[i], self.ciclos_volumen_integrado[i], self.Color_Combo.get())

    self.ax_presion_volumen.plot(aux_t, aux_t * self.pendientes[0] + (self.ordenada_origen[0]), 'k--')

```

```

self.ax_presion_volumen.set_ylabel('Volumen[mL]')
self.ax_presion_volumen.set_xlabel('Presión[kPa]')
self.ax_presion_volumen.set_title('Volumen integrado vs Presión sensada', **self.form_titulos)
self.ax_presion_volumen.legend()

self.canvas_flujo_tiempo.draw()

self.canvas_presion_tiempo.draw()
self.canvas_volumen_integrado_y_presion_tiempo.draw()
self.canvas_presion_volumen.draw()

self.toolbar_canvas_flujo_tiempo.update()
self.toolbar_presion_tiempo.update()
self.toolbar_volumen_integrado_y_presion_tiempo.update()
self.toolbar_presion_volumen.update()

self.body_text.grid(column=1, row=3, sticky=tk.NSEW, padx=5, pady=5)

self.texto.insert(tk.END, 'Las pendientes de ' + self.Lebel_var.get() + ' son: ' + str(self.pendientes) + ' ml/kPa\n')

self.texto.insert(tk.END, 'El promedio es: ')
self.promedio = 0
self.error_estandar = 0
self.desviacion_estandar = 0
self.error_estandar = 0
for element in self.pendientes:
    self.promedio = self.promedio + element
self.promedio = self.promedio / self.n_ciclo
self.texto.insert(tk.END, str(self.promedio) + ' ml/kPa\n')

self.texto.insert(tk.END, 'Los volumenes máximos de ' + self.Lebel_var.get() + ' son: ' + str(self.max_volumen) + ' ml\n')

for element in self.max_volumen:
    self.promedio_volumen = self.promedio_volumen + element
self.promedio_volumen = self.promedio_volumen / self.n_ciclo
self.texto.insert(tk.END, 'El promedio es: ')
self.texto.insert(tk.END, str(self.promedio_volumen) + ' ml\n')
self.texto.insert(tk.END, '-----\n')

def _leer(self):

    self.path = tk.filedialog.askdirectory(title = "Select A File")
    with open(self.path + '/Set.txt') as self.file:
        self.rows = csv.reader(self.file, delimiter=',')
        self.Set = {}
        for self.row in self.rows:
            if not self.row: # Saltea filas sin datos
                continue
            self.row = [func(val) for func, val in zip([str,str], self.row) ]
            self.Set[self.row[0]] = self.row[1]
    self.file.close()
    with open(self.path + '/Data.txt', 'r') as self.file:
        self.rows = csv.reader(self.file, delimiter=',')
        self.column = 0
        self.gData = [[],[],[],[],[],[]]
        for self.row in self.rows:
            self.aux = self.row
            for self.aux in self.row:
                for i in self.aux.replace('[ , ]', replace(']','')).split(","):
                    self.gData[self.column].append(float(i))
            self.column = self.column+1
    self.file.close()
    self.create_body_plot()

def _agregar(self):

    self.path = tk.filedialog.askdirectory(title = "Select A File")
    with open(self.path + '/Set.txt') as self.file:
        self.rows = csv.reader(self.file, delimiter=',')

```

```

    self.Set={}
    for self.row in self.rows:
        if not self.row: # Saltea filas sin datos
            continue
        self.row = [func(val) for func, val in zip([str,str], self.row) ]
        self.Set[self.row[0]]=self.row[1]
    self.file.close()
with open(self.path + '/Data.txt', 'r') as self.file:
    self.rows = csv.reader(self.file,delimiter=',')
    self.column = 0
    self.gData=[[[],[],[],[],[],[]]]
    for self.row in self.rows:
        self.aux = self.row
        for self.aux in self.row:
            for i in self.aux.replace('[','').replace(']', '').split(","):
                self.gData[self.column].append(float(i))
            self.column = self.column+1
    self.file.close()
    self.agregar_plot()

def _salvar(self):
    self.path= tk.filedialog.asksaveasfilename(title = "Select A File", filetypes = ((("All Files", "*.*"), ("Python files", "*.png"), ("mp4", ".mp4"), ("jpg", ".jpg"))))
    os.mkdir(self.path)
    with open(self.path+'/Data.txt', 'w') as self.file:
        writer = csv.writer(self.file,delimiter=',')
        writer.writerow(self.gData)
        self.file.close()
    with open(self.path+'/Set.txt', 'w') as self.file:
        writer = csv.writer(self.file,delimiter=',')
        writer.writerow(['SegundosMostrados', str(self.Set['SegundosMostrados'])])
        writer.writerow(['TiempoMuestreo', str(self.Set['TiempoMuestreo'])])
        writer.writerow(['Puerto', str(self.Set['Puerto'])])
        writer.writerow(['VolumenArduino', str(self.Set['VolumenArduino'])])
        writer.writerow(['VelocidadInicial', str(self.Set['VelocidadInicial'])])
        writer.writerow(['VelocidadFinal', str(self.Set['VelocidadFinal'])])
        writer.writerow(['Cantidad', str(self.Set['Cantidad'])])
        writer.writerow(['TiempoPaso', str(self.Set['TiempoPaso'])])
        writer.writerow(['DeltaPaso', str(self.Set['DeltaPaso'])])
    self.file.close()
    self.fig_flujo_presion_steps_volumen.savefig(self.path+'/fig_flujo_presion_steps_volumen.jpg')
    self.fig_resistencia_compliance.savefig(self.path+'/resistencia_compliance.jpg')
    self.fig_resistencia_compliance_sen.savefig(self.path+'/resistencia_compliance_sen.jpg')

def _leer_set(self):
    try:
        with open('Set/Set.csv') as self.file:
            self.rows = csv.reader(self.file)
            self.Set={}
            for self.row in self.rows:
                if not self.row: # Saltea filas sin datos
                    continue
                self.row = [func(val) for func, val in zip([str,str], self.row) ]
                self.Set[self.row[0]]=self.row[1]
            self.file.close()

    except:
        showerror(title='Error', message="Error al leer el Set")
        #self._quit()

def _guardar_set(self):
    try:
        with open('Set/Set.csv', 'w') as self.file:
            writer = csv.writer(self.file,delimiter=',')
            writer.writerow(['SegundosMostrados', str(self.Set['SegundosMostrados'])])
            writer.writerow(['TiempoMuestreo', str(self.Set['TiempoMuestreo'])])
            writer.writerow(['Puerto', str(self.Set['Puerto'])])
            writer.writerow(['VolumenArduino', str(self.Set['VolumenArduino'])])
            writer.writerow(['VelocidadInicial', str(self.Set['VelocidadInicial'])])
            writer.writerow(['VelocidadFinal', str(self.Set['VelocidadFinal'])])
            writer.writerow(['Cantidad', str(self.Set['Cantidad'])])
            writer.writerow(['TiempoPaso', str(self.Set['TiempoPaso'])])
            writer.writerow(['DeltaPaso', str(self.Set['DeltaPaso'])])
        self.file.close()

    except:
        showerror(title='Error',
                  message="Error al guardar el Set")

```

```
def _quit(self):
    self.quit()      # stops mainloop
    self.destroy()   # this is necessary on Windows to prevent
                    # Fatal Python Error: PyEval_RestoreThread: NULL tstate

App = Entorno()
App.mainloop()
```

Firmware del Arduino Uno

Este programa se encuentra publicado en un repositorio libre de git-hub, (<https://github.com/ifenoy/Lung-function>).

```

#include <Wire.h>

class SFM3000CORE {
public:
    //SFM3000CORE(uint8_t i2cAddress);
    SFM3000CORE(int i2cAddress);
    void init();
    int getvalue();

private:
    //uint8_t mI2cAddress;
    int mI2cAddress;
    uint8_t crc8(const uint8_t data, uint8_t crc);
};

//variables globales
const int dirPin = 3;
const int stepPin = 4;
const int sleepPin = 5;
const int enablePin = 7;
const int resetPin = 6;

const int sensorPresionPositiva = A0;
#define offsetPresionPositiva 34.5
#define proporcionalPresionPositiva 92.2

const int sensorPresionNegativa = A1;
#define offsetPresionNegativa 38.5
#define proporcionalPresionNegativa 91.9

long executing_time_us = 0; //
long executing_steps_time_us = 0; //
long executing_muestreo_time_us = 0; //

int StepsActual=0;
int StepsInspiracion=0;
int StepsEspiracion=0;

float Volumen=0;
unsigned long TiempoInicial=0;
long TiempoDelta=0;

int NumeroCiclo = 1;
int CantidadCiclos = 0;

unsigned long previousMicro=0;
unsigned long currentMicros=0;

unsigned long Ts_Inspiracion=0;
unsigned long Ts_Espiracion=0;

unsigned long Ts_Muestreo=5000;

int esperaFinCiclo = 0;

float auxPresionPositiva = 0;
float auxPresionNegativa = 0;

float presionMax = 0;

boolean stepsState=LOW;
boolean imprimirEstados = LOW;

SFM3000CORE senseFlow(64);

void enviar(float flujo, float presionNegativa, float presionPositiva, float steps)
{
    Serial.print(String(flujo)+",");
    Serial.print(String(presionNegativa)+",");
    Serial.print(String(presionPositiva)+",");
    Serial.println(String(steps));
}

void dar_paso(int dir)
{

```

```

stepsState = !stepsState;
digitalWrite(dirPin, dir);
digitalWrite(stepPin, stepsState);
}

enum State
{
    Inicio,
    Inspiracion,
    Espiracion,
    FinCiclo,
    Espera,
    NuevosParametros
};

State EstadoActual=Inicio;

void estadoInicio()
{
    if (imprimirEstados)
    { Serial.println("estadoInicio");}
    EstadoActual=Espera;
}

void estadoEspera()
{
    if(Serial.available())
    {EstadoActual=NuevosParametros;
     if (imprimirEstados)
    {
        Serial.println("NuevosParametros"); } }
}

void estadoInspiracion()
{
    if ((currentMicros - executing_muestreo_time_us) >= Ts_Muestreo)
    {
        executing_muestreo_time_us = currentMicros;
        float auxFlujo=float(senseFlow.getvalue())/50.0;
        auxPresionNegativa = (float(analogRead(sensorPresionNegativa))- offsetPresionNegativa)/proporcionalPresionNegativa ;
        auxPresionPositiva = (float(analogRead(sensorPresionPositiva))- offsetPresionPositiva)/proporcionalPresionPositiva ;
        float auxVolumen = 0;
        if (StepsActual>50)
        {
            auxVolumen = float(StepsActual -50)/135.0;
        }
        enviar(auxFlujo,auxPresionNegativa,auxPresionPositiva,auxVolumen); //envio sensores y pasos al python
    }

    if ((currentMicros - executing_steps_time_us) >= Ts_Inspiracion) //tiempo entre pasos de inspiracion
    {
        executing_steps_time_us = currentMicros;
        //Ts_Inspiracion = Ts_Inspiracion + 2;
        StepsActual++;
        dar_paso(HIGH); //doy un paso
    }

    if(StepsActual>=StepsInspiracion) //termina la inspiracion arranca la espiracion
    {
        if (imprimirEstados)
        { Serial.println("estadoEspiracion");}
        EstadoActual=Espiracion;
    }

    if(auxPresionPositiva >=presionMax) //termina la inspiracion arranca la espiracion
    {
        if (imprimirEstados)
        { Serial.println("estadoEspiracion");}
        Ts_Inspiracion = StepsActual;
        EstadoActual=Espiracion;
    }
}

void estadoEspiracion()
{
    if ((currentMicros - executing_muestreo_time_us) >= Ts_Muestreo)
}

```

```

{
    executing_muestreo_time_us = currentMicros;
    float auxFlujo=float(senseFlow.getvalue())/50.0;
    auxPresionNegativa = (float(analogRead(sensorPresionNegativa))- offsetPresionNegativa)/proporcionalPresionNegativa ;
    auxPresionPositiva = (float(analogRead(sensorPresionPositiva))- offsetPresionPositiva)/proporcionalPresionPositiva ;
    float auxVolumen = float(StepsInspiracion -50)/135.0;
    if (StepsInspiracion -StepsActual >50)
    {
        auxVolumen = float(StepsActual)/135.0;
    }
    enviar(auxFlujo ,auxPresionNegativa ,auxPresionPositiva ,auxVolumen); //envio sensores y pasos al python
}

if(auxPresionNegativa>=presionMax) //termina la inspiracion arranca la espiracion
{
    if (imprimirEstados)
    {Serial.println("estadoEspiracion");}
    EstadoActual=Espera;
}

if ((currentMicros - executing_steps_time_us) >= Ts_Espiracion) //tiempo entre pasos de inspiracion
{
    executing_steps_time_us = currentMicros;
    //Ts_Espiracion = Ts_Espiracion + 2;
    StepsActual--;
    dar_paso(HIGH); //doy un paso
}

if(StepsActual<0) //termina la espiracion termina el ciclo
{EstadoActual=FinCiclo;
esperaFinCiclo = 0;}
}

void estadoFinCiclo()
{
    if (imprimirEstados)
    {
    Serial.println("Finciclo"); }

    if ((currentMicros - executing_muestreo_time_us) >= Ts_Muestreo)
    {
        executing_muestreo_time_us = currentMicros;
        float auxFlujo=float(senseFlow.getvalue())/50.0;
        auxPresionNegativa = (float(analogRead(sensorPresionNegativa))- offsetPresionNegativa)/proporcionalPresionNegativa ;
        auxPresionPositiva = (float(analogRead(sensorPresionPositiva))- offsetPresionPositiva)/proporcionalPresionPositiva ;
        float auxStepsActual=0;
        enviar(auxFlujo ,auxPresionNegativa ,auxPresionPositiva ,auxStepsActual); //envio sensores y pasos al python
    }

    if ((currentMicros - executing_steps_time_us) >= Ts_Espiracion) //tiempo entre pasos de inspiracion
    {
        executing_steps_time_us = currentMicros;
        esperaFinCiclo=esperaFinCiclo +1;
    }

    if(Serial.available ())
    {EstadoActual=NuevosParametros;}
    if (esperaFinCiclo >=300)
    {

        Ts_Inspiracion = TiempoInicial;
        Ts_Espiracion = TiempoInicial;
        if (imprimirEstados)
        {Serial.println("estadoInspiracion");}
        EstadoActual=Inspiracion;
    }
}

void estadoNuevosParametros()
{
    String cad = Serial.readString();
    if (imprimirEstados)
    {
    Serial.println(cad);}
    int pos = cad.indexOf(',');
    String cad1= cad.substring(0,pos);
    cad= cad.substring(pos+1);
    pos = cad.indexOf(',');
    String cad2= cad.substring(0,pos);
    cad= cad.substring(pos+1);
}

```

```

pos = cad.indexOf(',');
String cad3= cad.substring(0,pos);
cad= cad.substring(pos+1);
pos = cad.indexOf(',');
String cad4= cad.substring(0,pos);
cad= cad.substring(pos+1);
pos = cad.indexOf(',');
String cad5= cad.substring(pos+1);

if (cad1.compareTo("Set")){} else{
    Volumen = cad2.toFloat();
    TiempoInicial = cad3.toFloat();
    presionMax = cad4.toFloat();
    CantidadCiclos = cad5.toFloat();
}
StepsInspiracion = (Volumen*135)+50; // relacion entre volumen ml y pasos
presionMax = presionMax;
Ts_Inspiracion = (TiempoInicial);
Ts_Espiracion =(TiempoInicial);

if (imprimirEstados)
{
Serial.println(Volumen);
Serial.println(StepsInspiracion);
Serial.println(TiempoInicial);
Serial.println(presionMax);
Serial.println(CantidadCiclos);
Serial.println(Ts_Inspiracion);
Serial.println(Ts_Espiracion);
}
if (Volumen==0)
{
    if (imprimirEstados)
    {Serial.println("espera");}
    digitalWrite(sleepPin ,LOW);
    digitalWrite(enablePin ,HIGH);
    EstadoActual=Espera ;} else
{
    if (imprimirEstados)
    {
        Serial.println("estadoInspiracion");
        digitalWrite(sleepPin ,HIGH);
        digitalWrite(enablePin ,LOW);
        EstadoActual=Inspiracion ;
    }
    StepsActual=0;
    NumeroCiclo = 1;
    executing_time_us = currentMicros;
    executing_steps_time_us = currentMicros;
}

void setup() {
    // put your setup code here, to run once:
senseFlow.init();

pinMode(13, OUTPUT);
pinMode(dirPin, OUTPUT);
pinMode(stepPin, OUTPUT);
pinMode(sleepPin, OUTPUT);
digitalWrite(sleepPin ,LOW);
pinMode(enablePin, OUTPUT);
digitalWrite(enablePin,HIGH);
pinMode(resetPin, OUTPUT);
digitalWrite(resetPin ,HIGH);

Serial.begin(115200);
}

void loop() {
    // put your main code here, to run repeatedly:
    currentMicros = micros();

    switch (EstadoActual)
    {
        case Inicio: estadoInicio(); break;
        case Inspiracion: estadoInspiracion(); break;
        case Espiracion: estadoEspiracion(); break;
        case FinCiclo: estadoFinCiclo(); break;
        case Espera: estadoEspera(); break;
    }
}

```

```

    case NuevosParametros: estadoNuevosParametros(); break;
}
}

SFM3000CORE::SFM3000CORE(int i2cAddress)
{
    //: mI2cAddress(i2cAddress)
    mI2cAddress = i2cAddress;
}

void SFM3000CORE::init()
{
    int a = 0;
    int b = 0;
    int c = 0;

    Wire.begin();
    //Serial.begin(9600);
    delay(1000);
    Wire.beginTransmission(byte(mI2cAddress)); // transmit to device with I2C mI2cAddress
    Wire.beginTransmission(byte(mI2cAddress)); // transmit to device with I2C mI2cAddress
    Wire.write(byte(0x10));      //
    Wire.write(byte(0x00));      //
    Wire.endTransmission();
    delay(5);

    Wire.requestFrom(mI2cAddress, 3); //
    a = Wire.read(); // received first byte stored here
    b = Wire.read(); // received second byte stored here
    c = Wire.read(); // received third byte stored here

    Wire.endTransmission();
    //Serial.print(a);
    //Serial.print(b);
    //Serial.println(c);

    delay(5);

    Wire.requestFrom(mI2cAddress, 3); //
    a = Wire.read(); // received first byte stored here
    b = Wire.read(); // received second byte stored here
    c = Wire.read(); // received third byte stored here
    Wire.endTransmission();
    //Serial.print(a);
    //Serial.print(b);
    //Serial.println(c);

    delay(5);
}

int SFM3000CORE::getvalue()
{
    Wire.requestFrom(mI2cAddress, 3); // set read 3 bytes from device with address 0x40
    uint16_t a = Wire.read(); // received first byte stored here. The variable "uint16_t" can hold 2 bytes, this will be relevant later
    uint8_t b = Wire.read(); // second received byte stored here
    uint8_t crc = Wire.read(); // crc value stored here
    uint8_t mycrc = 0xFF; // initialize crc variable
    mycrc = crc8(a, mycrc); // let first byte through CRC calculation
    mycrc = crc8(b, mycrc); // and the second byte too
    if (mycrc != crc) { // check if the calculated and the received CRC byte matches
        //Serial.println("Error: wrong CRC");
    }
    a = (a << 8) | b; // combine the two received bytes to a 16bit integer value
    // a >>= 2; // remove the two least significant bits
    //float Flow = (float)a;
    int flow = (int(a)+ 32768);
    //int flow = (result - offset) / scale;
    return flow;
}

uint8_t SFM3000CORE::crc8(const uint8_t data, uint8_t crc)
{
    crc ^= data;
    for (uint8_t i = 8; i; --i) {
        crc = (crc & 0x80)
            ? (crc << 1) ^ 0x31
            : (crc << 1);
    }
    return crc;
}

```


Anexo hoja de Datos

MPX5-010DP (NXP Semiconductors, EE. UU.)



Integrated Silicon Pressure Sensor On-Chip Signal Conditioned, Temperature Compensated and Calibrated

The MPX5700 series piezoresistive transducer is a state-of-the-art monolithic silicon pressure sensor designed for a wide range of applications, but particularly those employing a microcontroller or microprocessor with A/D inputs. This patented, single element transducer combines advanced micromachining techniques, thin-film metallization, and bipolar processing to provide an accurate, high level analog output signal that is proportional to the applied pressure.

Features

- 2.5% Maximum Error over 0° to 85°C
- Ideally Suited for Microprocessor or Microcontroller-Based Systems
- Available in Differential and Gauge Configurations
- Patented Silicon Shear Stress Strain Gauge
- Durable Epoxy Unibody Element

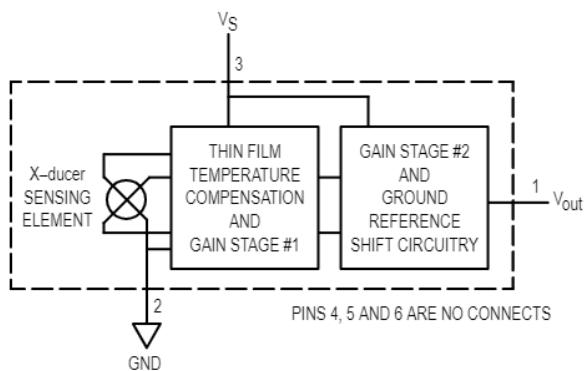


Figure 1. Fully Integrated Pressure Sensor Schematic

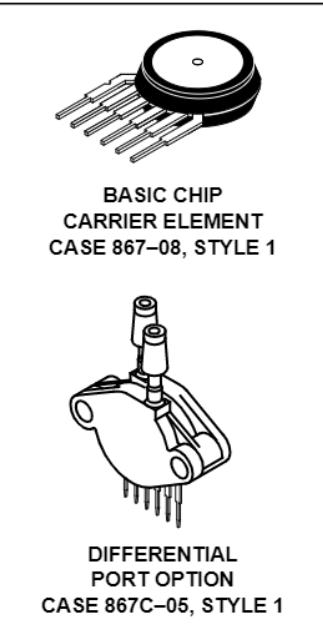
MAXIMUM RATINGS(1)

Parametrics	Symbol	Value	Unit
Overpressure ($P_2 \leq 1$ Atmosphere)	$P_{1\max}$	2800	kPa
Burst Pressure ($P_2 \leq 1$ Atmosphere)	$P_{1\text{burst}}$	5000	kPa
Storage Temperature	T_{stg}	-40 to +125	°C
Operating Temperature	T_A	-40 to +125	°C

1. $T_C = 25^\circ\text{C}$ unless otherwise noted. Maximum Ratings apply to Case 867-08 only.
2. Extended exposure at the specified limits may cause permanent damage or degradation to the device.
3. This sensor is designed for applications where P_1 is always greater than, or equal to P_2 .

MPX5700 SERIES

**INTEGRATED
PRESSURE SENSOR**
0 to 700 kPa (0 to 101.5 psi)
0.2 to 4.7 V OUTPUT



PIN NUMBER			
1	V_{out}	4	N/C
2	Gnd	5	N/C
3	V_S	6	N/C

NOTE: Pins 4, 5, and 6 are internal device connections. Do not connect to external circuitry or ground. Pin 1 is noted by the notch in the Lead.

MPX5700 SERIES

OPERATING CHARACTERISTICS ($V_S = 5.0$ Vdc, $T_A = 25^\circ\text{C}$ unless otherwise noted, $P_1 > P_2$)

Characteristic	Symbol	Min	Typ	Max	Unit
Pressure Range ⁽¹⁾	POP	0	—	700	kPa
Supply Voltage ⁽²⁾	V_S	4.75	5.0	5.25	Vdc
Supply Current	I_O	—	7.0	10	mAdc
Zero Pressure Offset ⁽³⁾ (0 to 85°C)	V_{off}	0.088	0.2	0.313	Vdc
Full Scale Output ⁽⁴⁾ (0 to 85°C)	V_{FSO}	4.587	4.7	4.813	Vdc
Full Scale Span ⁽⁵⁾ (0 to 85°C)	V_{FSS}	—	4.5	—	Vdc
Accuracy ⁽⁶⁾ (0 to 85°C)	—	—	—	± 2.5	% V_{FSS}
Sensitivity	V/P	—	6.4	—	mV/kPa
Response Time ⁽⁷⁾	t_R	—	1.0	—	ms
Output Source Current at Full Scale Output	I_O+	—	0.1	—	mAdc
Warm-Up Time ⁽⁸⁾	—	—	20	—	ms

Decoupling circuit shown in Figure 4 required to meet electrical specifications.

MECHANICAL CHARACTERISTICS

Characteristic	Symbol	Min	Typ	Max	Unit
Weight, Basic Element (Case 867)	—	—	4.0	—	Grams
Cavity Volume	—	—	—	0.01	IN ³
Volumetric Displacement	—	—	—	0.001	IN ³

NOTES:

1. 1.0 kPa (kiloPascal) equals 0.145 psi.
2. Device is ratiometric within this specified excitation range.
3. Offset (V_{off}) is defined as the output voltage at the minimum rated pressure.
4. Full Scale Output (V_{FSO}) is defined as the output voltage at the maximum or full rated pressure.
5. Full Scale Span (V_{FSS}) is defined as the algebraic difference between the output voltage at full rated pressure and the output voltage at the minimum rated pressure.
6. Accuracy (error budget) consists of the following:
 - Linearity: Output deviation from a straight line relationship with pressure over the specified pressure range.
 - Temperature Hysteresis: Output deviation at any temperature within the operating temperature range, after the temperature is cycled to and from the minimum or maximum operating temperature points, with zero differential pressure applied.
 - Pressure Hysteresis: Output deviation at any pressure within the specified range, when this pressure is cycled to and from the minimum or maximum rated pressure, at 25°C .
 - T_C Span: Output deviation over the temperature range of 0° to 85°C , relative to 25°C .
 - T_C Offset: Output deviation with minimum rated pressure applied, over the temperature range of 0° to 85°C , relative to 25°C .
 - Variation from Nominal: The variation from nominal values, for Offset or Full Scale Span, as a percent of V_{FSS} , at 25°C .
7. Response Time is defined as the time for the incremental change in the output to go from 10% to 90% of its final value when subjected to a specified step change in pressure.
8. Warm-up is defined as the time required for the device to meet the specified output voltage after the pressure has been stabilized.
9. P_2 max is 500 kPa.

ON-CHIP TEMPERATURE COMPENSATION, CALIBRATION AND SIGNAL CONDITIONING

Figure 3 illustrates both the Differential/Gauge and the Absolute Sensing Chip in the basic chip carrier (Case 867). A fluorosilicone gel isolates the die surface and wire bonds from the environment, while allowing the pressure signal to be transmitted to the sensor diaphragm. (For use of the MPX5700D in a high pressure, cyclic application, consult the factory.)

The MPX5700 series pressure sensor operating characteristics, and internal reliability and qualification tests are based on use of dry air as the pressure media. Media, other than dry air, may have adverse effects on sensor performance and

long-term reliability. Contact the factory for information regarding media compatibility in your application.

Figure 4 shows a typical decoupling circuit for interfacing the sensor to the A/D input of a microprocessor. Proper decoupling of the power supply is recommended.

Figure 2 shows the sensor output signal relative to pressure input. Typical, minimum, and maximum output curves are shown for operation over a temperature range of 0° to 85°C using the decoupling circuit below. (The output will saturate outside of the specified pressure range.)

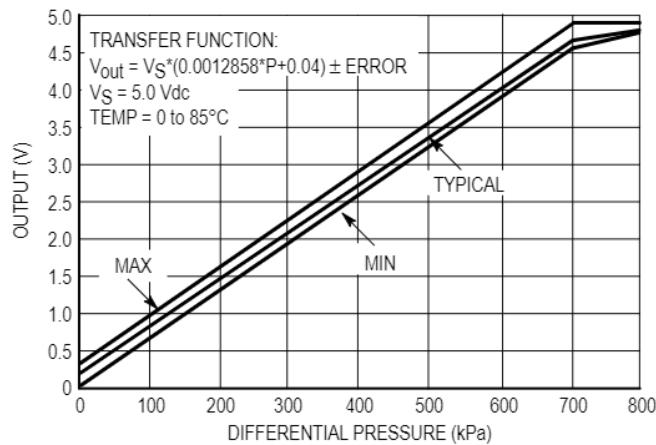


Figure 2. Output versus Pressure Differential

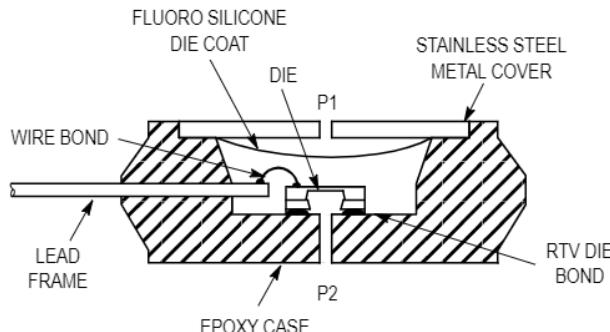


Figure 3. Cross-Sectional Diagram
(Not to Scale)

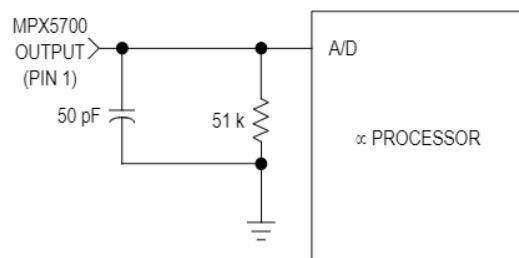


Figure 4. Typical Decoupling Filter for Sensor to
Microprocessor Interface

MPX5700 SERIES

PRESSURE (P1)/VACUUM (P2) SIDE IDENTIFICATION TABLE

Motorola designates the two sides of the pressure sensor as the Pressure (P1) side and the Vacuum (P2) side. The Pressure (P1) side is the side containing fluoro silicone gel which protects the die from harsh media. The Motorola MPX

pressure sensor is designed to operate with positive differential pressure applied, P1 > P2.

The Pressure (P1) side may be identified by using the table below:

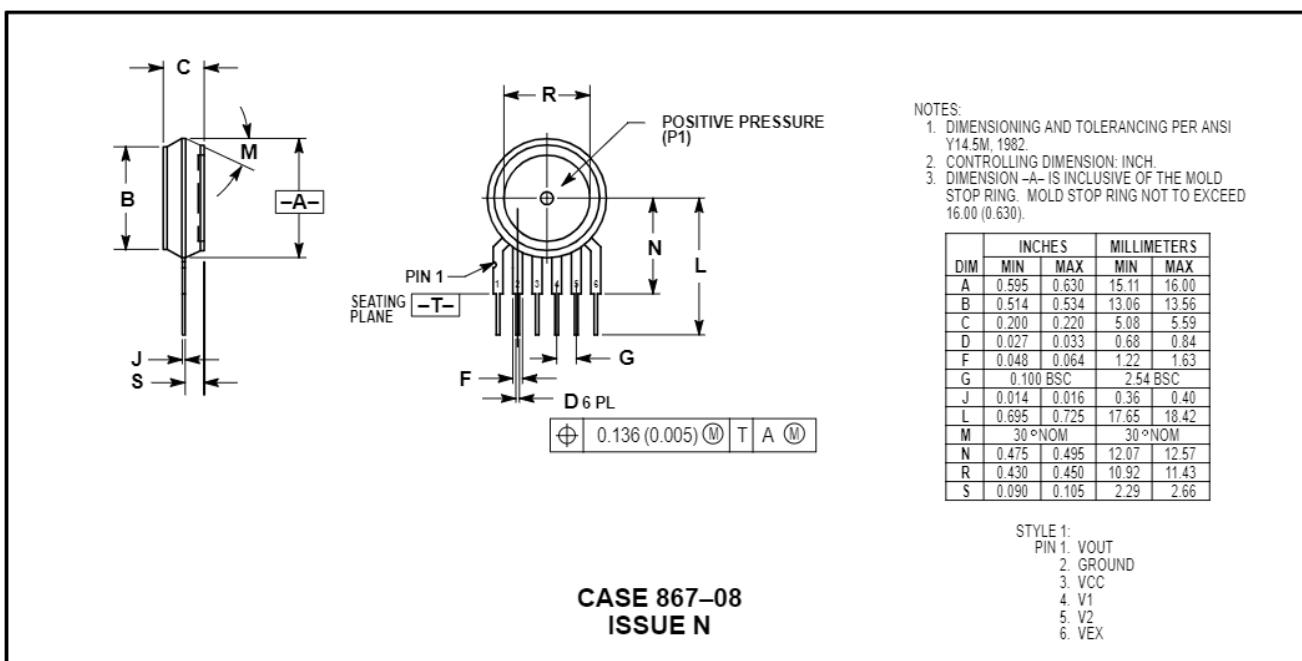
Part Number	Case Type	Pressure (P1) Side Identifier
MPX5700D	867-08	Stainless Steel Cap
MPX5700DP	867C-05	Side with Part Marking
MPX5700GP	867B-04	Side with Port Attached
MPX5700GS	867E-03	Side with Port Attached
MPX5700GSX	867F-03	Side with Port Attached

ORDERING INFORMATION

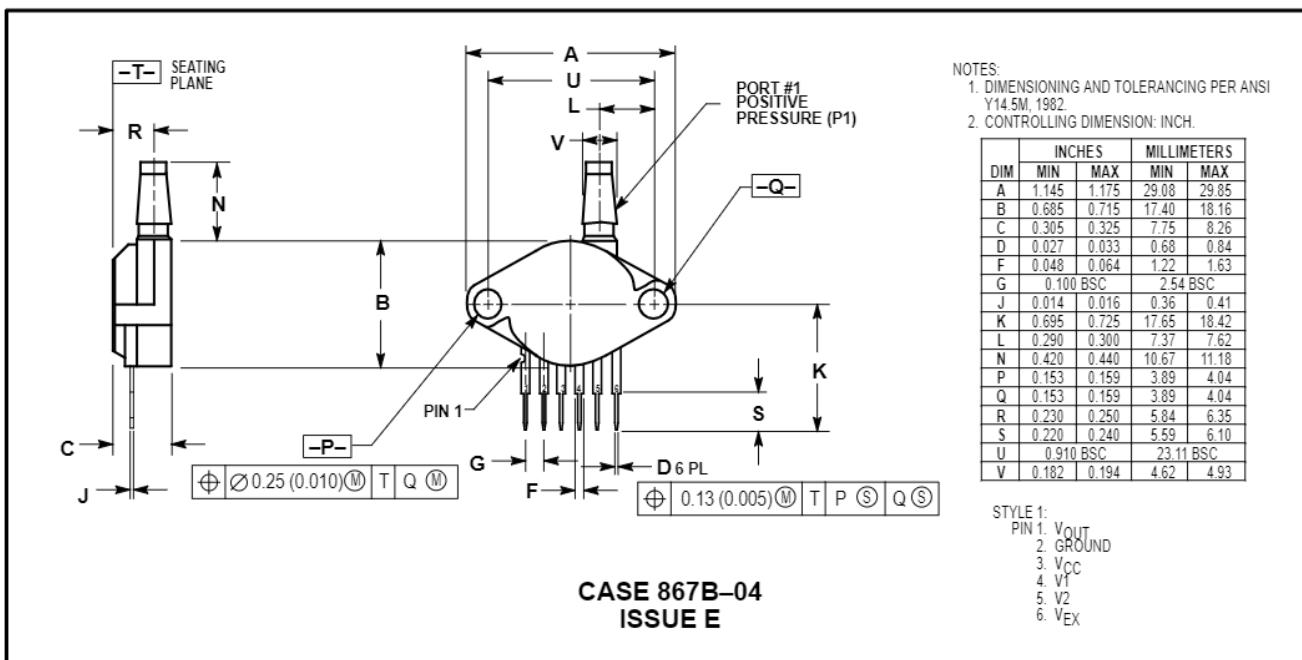
The MPX5700 pressure sensor is available in differential and gauge configurations. Devices are available in the basic element package or with pressure port fittings that provide printed circuit board mounting ease and barbed hose pressure connections.

Device Type	Options	Case Type	MPX Series	
			Order Number	Device Marking
Basic Element	Differential	867-08	MPX5700D	MPX5700D
Ported Elements	Differential Dual Ports	867C-05	MPX5700DP	MPX5700DP
	Gauge	867B-04	MPX5700GP	MPX5700GP
	Gauge, Axial	867E-03	MPX5700GS	MPX5700D
	Gauge, Axial PC Mount	867F-03	MPX5700GSX	MPX5700D

PACKAGE DIMENSIONS



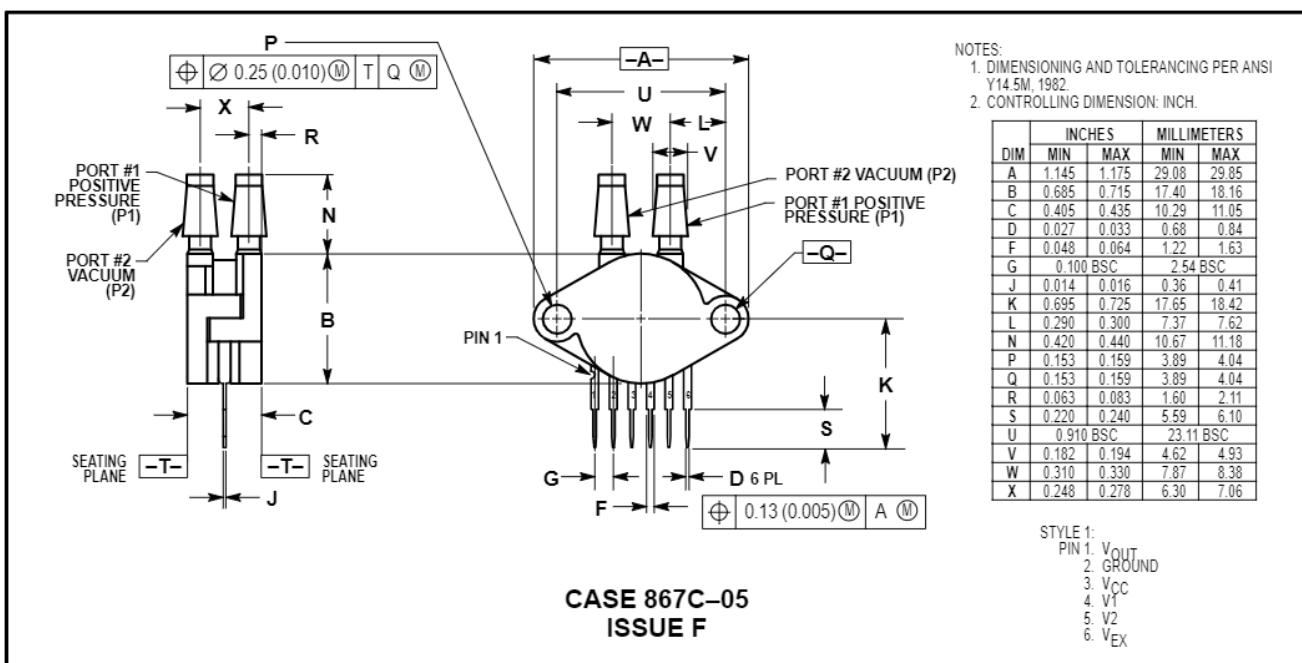
BASIC ELEMENT (A, D)



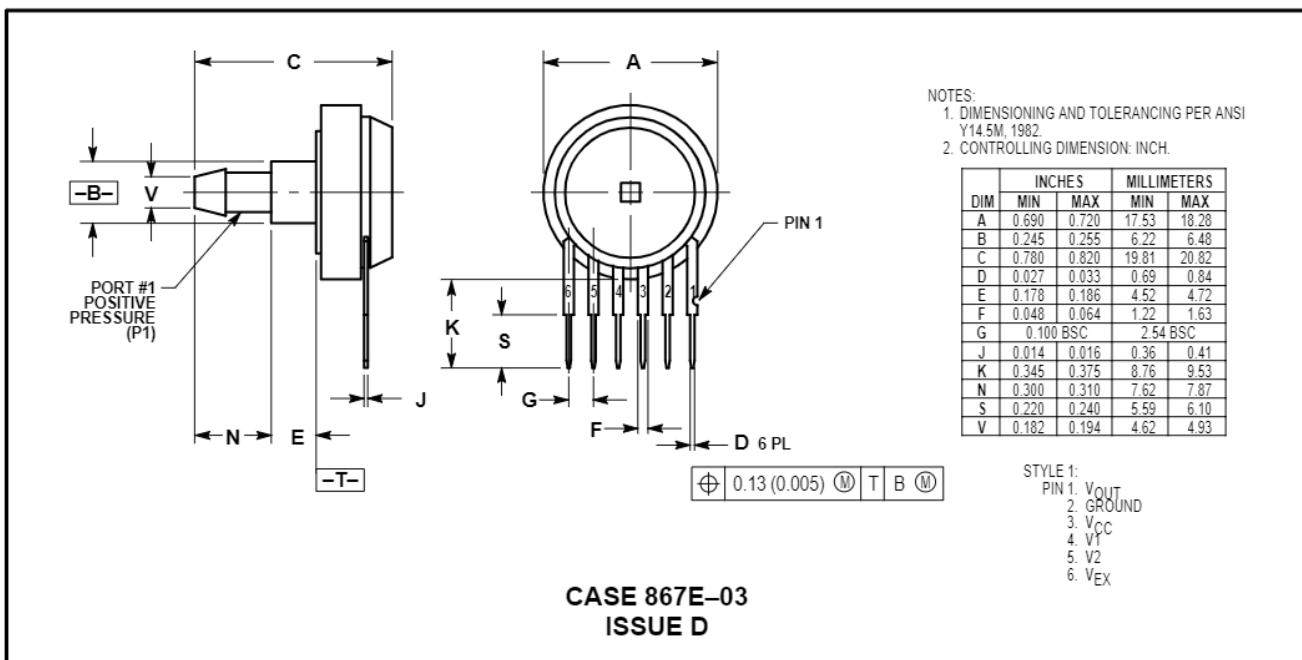
PRESSURE SIDE PORTED (AP, GP)

MPX5700 SERIES

PACKAGE DIMENSIONS—CONTINUED

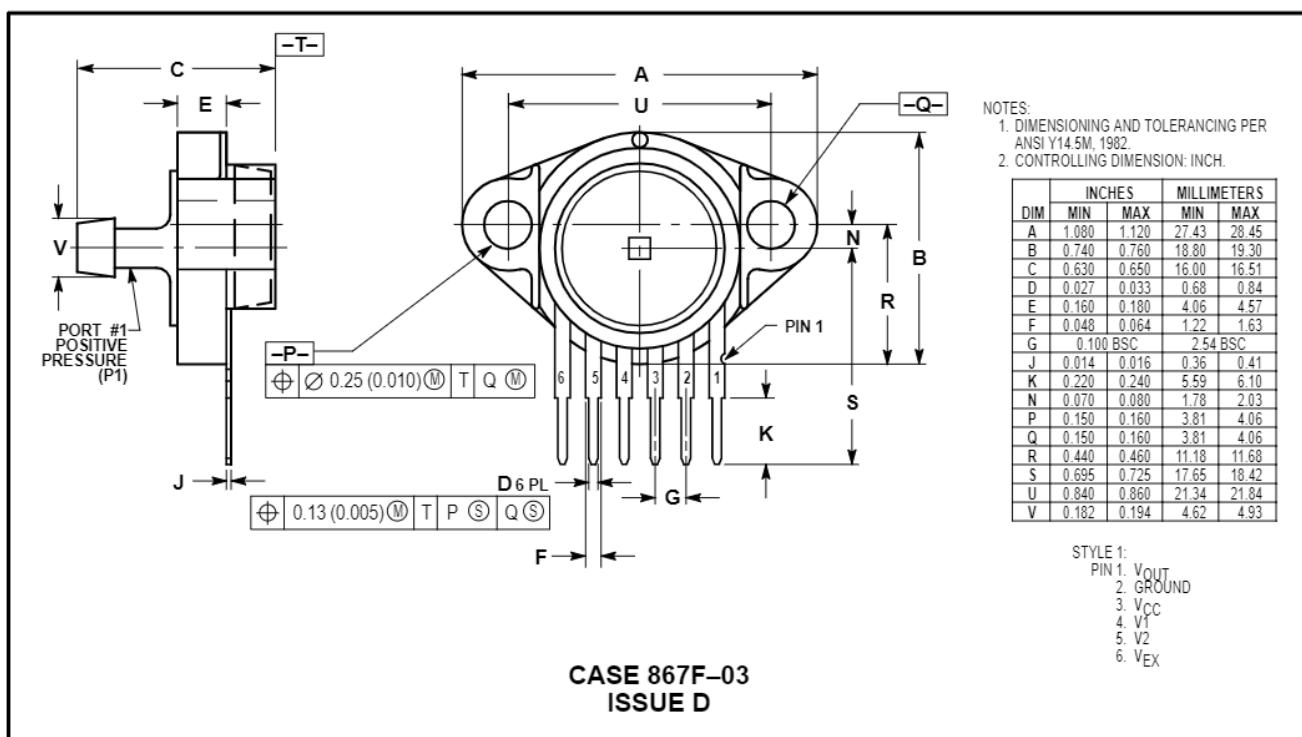


PRESSURE AND VACUUM SIDES PORTED (DP)



PRESSURE SIDE PORTED (AS, GS)

PACKAGE DIMENSIONS—CONTINUED



PRESSURE SIDE PORTED (ASX, GSX)

MPX5700 SERIES

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

Mfax is a trademark of Motorola, Inc.

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217. 303-675-2140 or 1-800-441-2447

JAPAN: Nippon Motorola Ltd.; Tatsumi-SPD-JLDC, 6F Seibu-Butsuryu-Center,
3-14-2 Tatsumi Koto-Ku, Tokyo 135, Japan. 81-3-3521-8315

Mfax™: RMFAX0@email.sps.mot.com – TOUCHTONE 602-244-6609
– US & Canada ONLY 1-800-774-1848
INTERNET: <http://motorola.com/sps>

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park,
51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298



FM3400-33-D (Sensirion, Suiza)

Datasheet SFM3400-AW & SFM3400-D

Digital Flow Meter for Neonatal/Pediatric Medical Applications

- Flow range: $\pm 33\text{slm}$ (bidirectional)
- Small dead space < 1ml
- Single-use (-D) and Re-use (-AW) version
- Very fast update time (0.5ms)



Product Summary

The SFM3400 sensor series is Sensirion's digital flow meter designed for neonatal/pediatric medical applications. It measures the flow rate of **air, oxygen and other non-aggressive gases** with superb accuracy. The special design of the flow channel results in a **very small dead space**.

Customers have the choice between the single-use SFM3400-D and the multi-use SFM3400-AW with the **ability to withstand autoclave procedures**. Therefore, the SFM3400 series is extremely well suited for proximal flow measurements in neonatal/pediatric medical ventilation and other respiratory applications.

The SFM3400 series has been designed with the use by medical professionals in mind. It features **medical cones** for pneumatic connection to standard breathing circuits and a mechanical interface for **easy and reliable electrical reconnection**. The sensor element, signal processing and digital calibration are on a single microchip assuring **very fast signal processing time, best-in-class accuracy** and **superior robustness** to rough handling and adverse conditions.

The well-proven and patented **CMOSens® sensor technology** is perfectly suited for high-quality mass production and is the ideal choice for demanding and cost-sensitive OEM applications.

Applications

- Proximal flow measurement for infants / neonates
- Expiratory flow measurement for infants / neonates
- Ventilation & Anesthesia for infants / neonates
- Respiratory measurements for infants / neonates
- Metabolic Measurements for infants / neonates

Sensor chip

The SFM3400 flow meter features a fifth-generation silicon sensor chip. In addition to a thermal mass flow sensor element, the chip contains an amplifier, A/D converter, EEPROM memory, digital signal processing circuitry, and interface. Due to seamless integration of signal acquisition and processing on the single silicon die significant performance and cost benefits are achieved.

OEM options

A variety of custom options can be implemented for high-volume OEM applications (custom flow rates, calibration for other gases, different body form factor, disposable option etc.). Contact us for more information.

1.1 Physical Specifications¹

Parameter	Condition	Value		Unit
Flow Range		-33 ... +33		slm ²
		Typical	Max ³	
Accuracy ⁴	span <33 slm offset	±3 ⁵ ± 0.02 ⁵	±10 ⁶ ± 0.05	% m.v. ⁷ slm ²
Noise Level ^{4,8}	span <33slm offset	0.5% ⁹ 0.005 ⁹	3.5% 0.025	% m.v. ⁷ slm ²
Accuracy shift for deviation from reference temperature 25°C	span offset		1% 0.01	% m.v./ 10°C slm / 10°C
Resolution (14bit)	span offset	0.06 ⁹ 0.005 ⁹	0.2 0.01	% m.v. ⁷ slm ²
Pressure Drop	@ 5 slm @ 10 slm @ 25 slm	100 / 0.4 ⁹ 250 / 1.0 ⁹ 900 / 3.6 ⁹	150 / 0.6 300 / 1.2 1500 / 6.0	Pa / inH ₂ O Pa / inH ₂ O Pa / inH ₂ O

1.2 Ambient conditions

Parameter	Condition	Value	Unit
Calibrated Temperature Range	dry gas	+10 ... +50	°C
Operating Temperature Range ¹⁰	10-95% rel. hum. (non cond.)	+5 ... +50	°C
Storage Temperature	10-95% rel. hum. (non cond.)	-40 ... +70	°C
Shelf Life for SFM3400-D	15°C - 35°C; 30 - 70 % rel. hum. storage in original packaging	3	years
Operating Pressure Range	absolute	0.54 – 1.1	bar
Burst Overpressure	gauge	0.3	bar

1.3 Media compatibility

Parameter	Value
Calibration ¹¹	Air
Media Compatibility	Air (non-condensing), N ₂ , O ₂ , other non-aggressive gases
Wetted Materials -AW	Si, Si ₃ N ₄ , SiO _x , gold, epoxy, PPSU, silicone
Wetted Materials -D	Si, Si ₃ N ₄ , SiO _x , gold, epoxy, MABS, silicone
RoHS, REACH	RoHS and REACH compliant

¹ Reference conditions are temperature = 25°C, absolute pressure = 966 mbar, horizontal flow and Vdd = 5V

² slm: mass flow measured in liters per minute at standard conditions (T = 20 °C, p = 1013.25 mbar)

³ For "Max" no sensor measured outside of this limit will be shipped and a CpK of 1.33 is targeted

⁴ For accuracy, noise level or resolution the total value is the sum of the offset and span values

⁵ This value corresponds to a CpK of 0.67 (95% of sensors within the "Typical" limit)

⁶ Dependent on tracheal tube diameter

⁷ % m.v. = % measured value = % of reading

⁸ Noise level defined as standard deviation of individual sensor readings, measured at full sampling rate

⁹ Average value

¹⁰ Do not exceed these operating conditions by heating the sensor excessively with the external heater, see section 4.4.

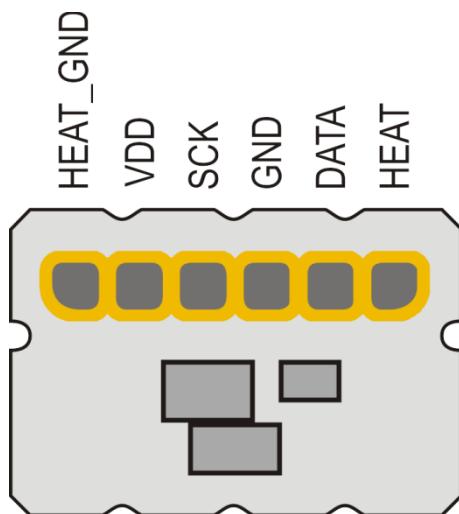
¹¹ Contact Sensirion for information about other gases, wider calibrated temperature ranges and higher storage temperatures

2. Electrical Specifications

2.1 Electrical Characteristics

Electrical properties	Condition	Value	Unit	
Interface		I ² C		
Default Sensor Address		64 (h40)		
Update Time	14 bit	0.5	ms	
Soft Reset Time		80	ms	
Start-up Time ¹²	Max.	100	ms	
I ² C bus Clock Frequency	Max.	400	kHz	
Supply Voltage		5V±5%	V	
Communication Level	High Low	Min. 2.5 GND	Max. VDD 1.1	V
Power Consumption ¹³		< 50	mW	
Electrical Connector		See section 2.2 and 3.2		
External Heater Power Rating ¹⁴	Max.	0.5	W	
External Heater Resistance ¹⁴	Typ.	51	Ω	
Output Signal Resolution ¹⁵		14	bit	
Scale Factor Flow	Air, N2	800	1/slm	
Offset Flow		32768		

2.2 Pad Layout



2.3 Conversion to Physical Values

In order to obtain the measured flow in [slm], the measured value needs to be converted using the following formula:

$$\text{flow [slm]} = \frac{\text{measured value} - \text{offset flow}}{\text{scale factor flow}}$$

Please note that the first measurement performed directly after chip initialization is not valid.

¹² After 4.75V is reached

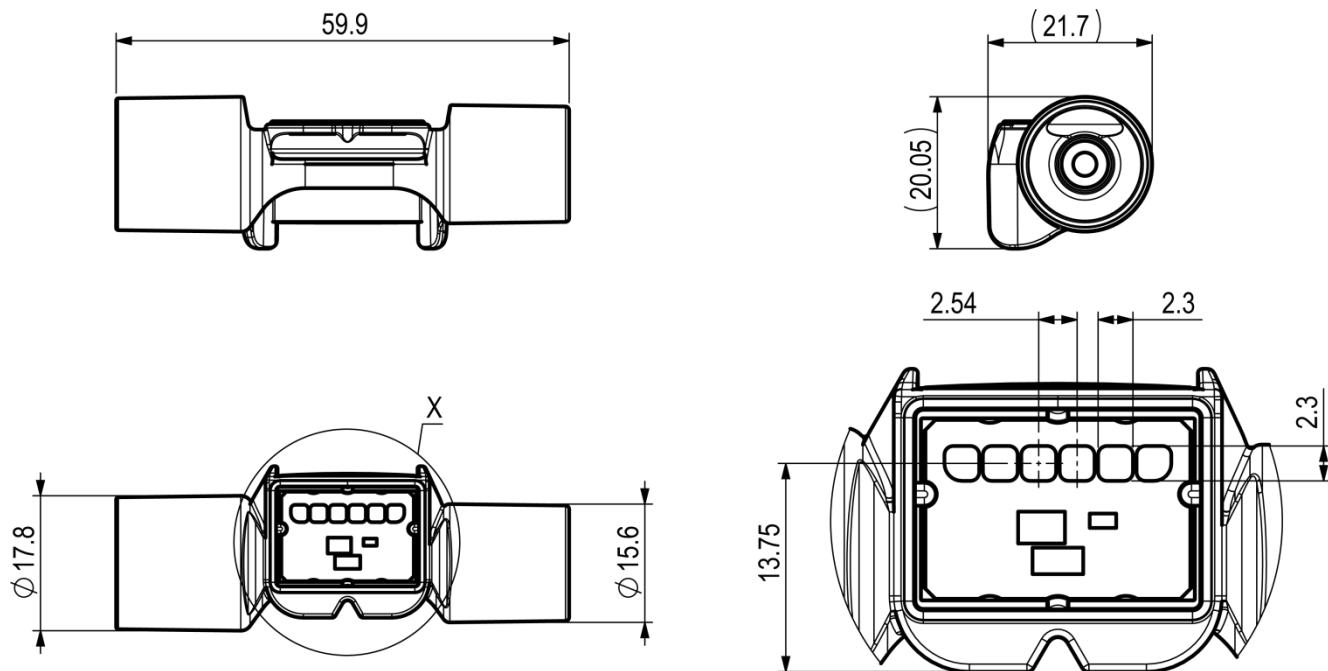
¹³ When the heater resistor on the PCB is not in operation

¹⁴ The heater's purpose is to avoid condensation or icing. One should only apply sufficient power to achieve this purpose. See section 4.4 for more details.

¹⁵ 16 bit with two least significant bits always zero

3. Mechanical Specifications

All dimensions are in millimeters (mm).



3.1 Mechanical fitting

Fittings of the SFM3400 sensor correspond to the international standard ISO5356-1:2004. Details about this type of connection can be found in the description of the standard.

3.2 Mechanical / Electrical Interface

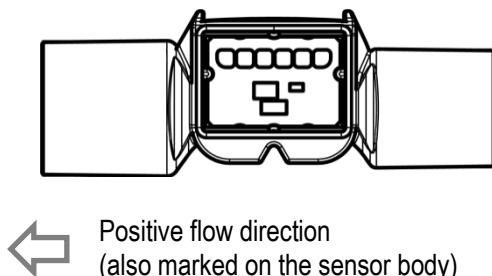
SFM3400 series has been designed for use in an expiratory environment. Therefore, the sensor has been designed for a connector that can be easily connected and disconnected. The connector itself is not provided as a standard product by Sensirion but Sensirion can help with an application note including design recommendations.

Dimension	Condition	Value
Length		60 mm (typical)
Diameter flow channel	center of sensor	3.5 mm (typical)
Medical connector, distal side	ISO 5356-1	Cone, 15 mm
Medical connector, proximal side	ISO 5356-1	Socket, 15 mm
Dead space	relevant section	0.9 ml (typical)
Weight	without connector	7.5 g (typical)

4. Instructions for Use

4.1 Calibration orientation

The sensors are calibrated horizontally as depicted in the following graph:



Positive flow direction is from the male 15 mm cone to the female 15 mm medical cone, so that inspiratory flows are positive and expiratory flows negative.

4.2 Inlet flow conditions

In order to provide good flow conditions, the inner diameter of the connecting tube has to be approximately the same as the inner diameter of the SFM3400 main flow channel.

4.3 Temperature compensation

The SFM3400-AW sensor features digital temperature compensation. The temperature is measured on the CMOSens® chip by an on-chip temperature sensor. This data is fed to a compensation circuit that is also integrated on the CMOSens® sensor chip. Thus, no external temperature compensation is necessary.

4.4 Heater operation

The sensor has an external heater for cases where the gas flowing through the sensor contains high humidity and is warmer than the ambient temperature. In such cases, heating the sensor can help to avoid condensation or icing. Sufficient power should be provided for this purpose, but excessive power must be avoided for two reasons:

- 1) to stay within the operating temperature range,
- 2) to maintain the best accuracy.

Reading out the chip temperature can be used as feedback on the current sensor temperature.

4.5 Cleaning (applies only to SFM3400-AW version)

The SFM3400-AW has been designed to withstand medical cleaning procedures. For details of the test and the results contact Sensirion.

Sensirion AG does not guarantee the stability of the flow sensor using arbitrary methods and/or equipment for autoclaving. Validation of the flow sensor stability for a specific type of procedure and/or equipment is the sole responsibility of the customer.

4.6 ESD

The electronics of the SFM3x00-AW flow sensor chip has been tested for ESD and passed an 8kV contact test. For ESD information about the additional EEPROM (type: 24LC01BT-I/MC) please consult the datasheet of the EEPROM.

4.7 Sensor handling

The SFM3400 sensor is designed to be robust and shock resistant. Nevertheless, the accuracy of the high-precision SFM3400 can be degraded by rough handling. Sensirion does not guarantee proper operation in case of improper handling.

Note: never connect the sensor while connecting part is wet. Especially after cleaning procedure special care is needed to dry the sensor.

Please be aware that SFM3400 has been designed for usage with air and other non-corrosive and non-toxic gases.

If leak tightness is critical in customer application, it remains customer's sole responsibility to leak-test the sensor before usage.

4.8 I²C Interface and communication

Due to I²C interface restrictions, the cable length from the sensor to the microprocessor is recommended to be as short as possible and certainly not above 30 cm. For wires longer than 10 cm it is mandatory to shield the SDA and SCL.

In case data is read from the sensor, the first data byte of the transaction must always be acknowledged by the master.

It must be possible to reset the sensor through a hard reset, i.e. powering off and on the sensor, in case the sensor freezes.

I²C Communication details are given in the application note "GF_AN_SFM3xx_I²C Functional Description".

There is an additional EEPROM on the PCB to allow storage of customer-specific data. Please see all details in the datasheet of the EEPROM. The EEPROM is of type 24LC01BT-I/MC. No additional validation or modification of EEPROM settings has been performed by Sensirion.

There is an additional EEPROM on the SFM3400-AW to allow storage of customer-specific data (like for example usage hours). Please see all details in the datasheet of the

EEPROM. The EEPROM is of type 24LC01BT-I/MC. No additional validation or modification of EEPROM settings has been performed by Sensirion.

5. Ordering Information

Use the part names and product numbers shown in the table below when ordering SFM3400-AW sensors. For the latest product information and local distributors, visit www.sensirion.com.

Part name	Product Number	Availability
SFM3400-33-AW	1-101545-01	Available
SFM3400-33-D	1-101607-01	Samples

Packaging units: 30 items/tray.

Every sensor is traceable by a unique serial number.

Revision history

Date	Author	Version	Changes
January 2020	DAT	1.1	Release of SFM3400-D version;
February 2021	DAT	1.2	Minor correction, added missing epoxy to the wetted materials list of -AW version
January 2022	PSIM	1.3	Note on operating temperature conditions and heater's operation

Important Notices

Warning, personal injury

Do not use this product as safety or emergency stop devices or in any other application where failure of the product could result in personal injury (including death). Do not use this product for applications other than its intended and authorized use. Before installing, handling, using or servicing this product, please consult the datasheet and application notes. Failure to comply with these instructions could result in death or serious injury.

If the Buyer shall purchase or use SENSIRION products for any unintended or unauthorized application, Buyer shall defend, indemnify and hold harmless SENSIRION and its officers, employees, subsidiaries, affiliates and distributors against all claims, costs, damages and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if SENSIRION shall be allegedly negligent with respect to the design or the manufacture of the product.

ESD Precautions

The inherent design of this component causes it to be sensitive to electrostatic discharge (ESD). To prevent ESD-induced damage and/or degradation, take customary and statutory ESD precautions when handling this product.

Warranty

SENSIRION warrants solely to the original purchaser of this product for a period of 12 months (one year) from the date of delivery that this product shall be of the quality, material and workmanship defined in SENSIRION's published specifications of the product. Within such period, if proven to be defective, SENSIRION shall repair and/or replace this product, in SENSIRION's discretion, free of charge to the Buyer, provided that:

- notice in writing describing the defects shall be given to SENSIRION within fourteen (14) days after their appearance;
- such defects shall be found, to SENSIRION's reasonable satisfaction, to have arisen from SENSIRION's faulty design, material, or workmanship;

- the defective product shall be returned to SENSIRION's factory at the Buyer's expense; and
- the warranty period for any repaired or replaced product shall be limited to the unexpired portion of the original period.

This warranty does not apply to any equipment which has not been installed and used within the specifications recommended by SENSIRION for the intended and proper use of the equipment. EXCEPT FOR THE WARRANTIES EXPRESSLY SET FORTH HEREIN, SENSIRION MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THE PRODUCT. ANY AND ALL WARRANTIES, INCLUDING WITHOUT LIMITATION, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ARE EXPRESSLY EXCLUDED AND DECLINED.

SENSIRION is only liable for defects of this product arising under the conditions of operation provided for in the datasheet and proper use of the goods. SENSIRION explicitly disclaims all warranties, express or implied, for any period during which the goods are operated or stored not in accordance with the technical specifications.

SENSIRION does not assume any liability arising out of any application or use of any product or circuit and specifically disclaims any and all liability, including without limitation consequential or incidental damages. All operating parameters, including without limitation recommended parameters, must be validated for each customer's applications by customer's technical experts. Recommended parameters can and do vary in different applications.

SENSIRION reserves the right, without further notice, (i) to change the product specifications and/or the information in this document and (ii) to improve reliability, functions and design of this product.

Copyright © 2021, SENSIRION.
CMOSens® is a trademark of Sensirion
All rights reserved

Headquarters and Subsidiaries

SENSIRION AG
Laubisruetistr. 50
CH-8712 Staefa ZH
Switzerland

phone: +41 44 306 40 00
fax: +41 44 306 40 30
info@sensirion.com
www.sensirion.com

Sensirion AG (Germany)
phone: +41 44 927 11 66
info@sensirion.com
www.sensirion.com

Sensirion Inc., USA
phone: +1 805 409 4900
info_us@sensirion.com
www.sensirion.com

Sensirion Japan Co. Ltd.
phone: +81 3 3444 4940
info@sensirion.co.jp
www.sensirion.co.jp

Sensirion Korea Co. Ltd.
phone: +82 31 345 0031 3
info@sensirion.co.kr
www.sensirion.co.kr

Sensirion China Co. Ltd.
phone: +86 755 8252 1501
info@sensirion.com.cn
www.sensirion.com.cn

To find your local representative, please visit www.sensirion.com/contact

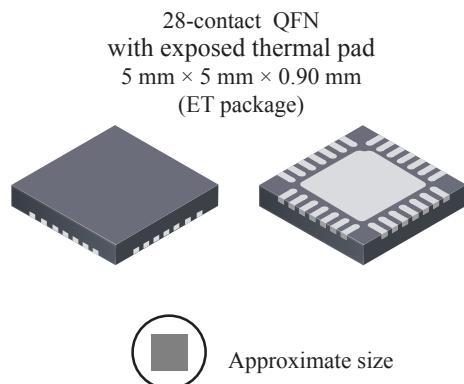
A4988 (Pololu, EE.UU.)

DMOS Microstepping Driver with Translator And Overcurrent Protection

Features and Benefits

- Low $R_{DS(ON)}$ outputs
- Automatic current decay mode detection/selection
- Mixed and Slow current decay modes
- Synchronous rectification for low power dissipation
- Internal UVLO
- Crossover-current protection
- 3.3 and 5 V compatible logic supply
- Thermal shutdown circuitry
- Short-to-ground protection
- Shorted load protection
- Five selectable step modes: full, $1/2$, $1/4$, $1/8$, and $1/16$

Package:



Description

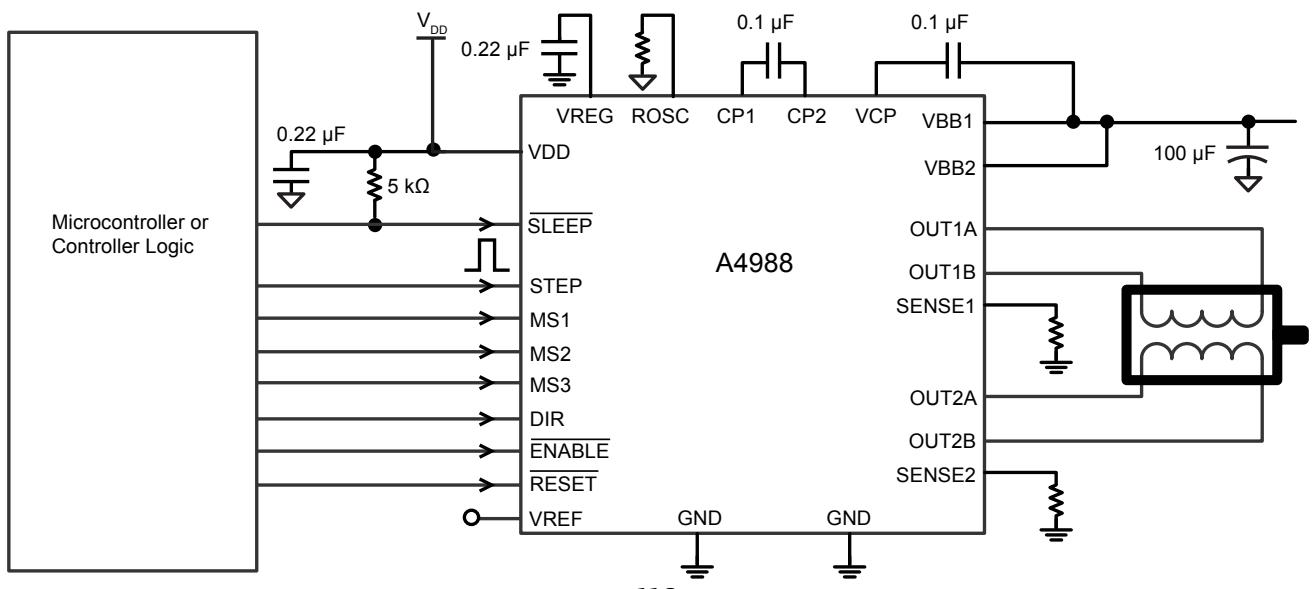
The A4988 is a complete microstepping motor driver with built-in translator for easy operation. It is designed to operate bipolar stepper motors in full-, half-, quarter-, eighth-, and sixteenth-step modes, with an output drive capacity of up to 35 V and ± 2 A. The A4988 includes a fixed off-time current regulator which has the ability to operate in Slow or Mixed decay modes.

The translator is the key to the easy implementation of the A4988. Simply inputting one pulse on the STEP input drives the motor one microstep. There are no phase sequence tables, high frequency control lines, or complex interfaces to program. The A4988 interface is an ideal fit for applications where a complex microprocessor is unavailable or is overburdened.

During stepping operation, the chopping control in the A4988 automatically selects the current decay mode, Slow or Mixed. In Mixed decay mode, the device is set initially to a fast decay for a proportion of the fixed off-time, then to a slow decay for the remainder of the off-time. Mixed decay current control results in reduced audible motor noise, increased step accuracy, and reduced power dissipation.

Continued on the next page...

Typical Application Diagram



Description (continued)

Internal synchronous rectification control circuitry is provided to improve power dissipation during PWM operation. Internal circuit protection includes: thermal shutdown with hysteresis, undervoltage lockout (UVLO), and crossover-current protection. Special power-on sequencing is not required.

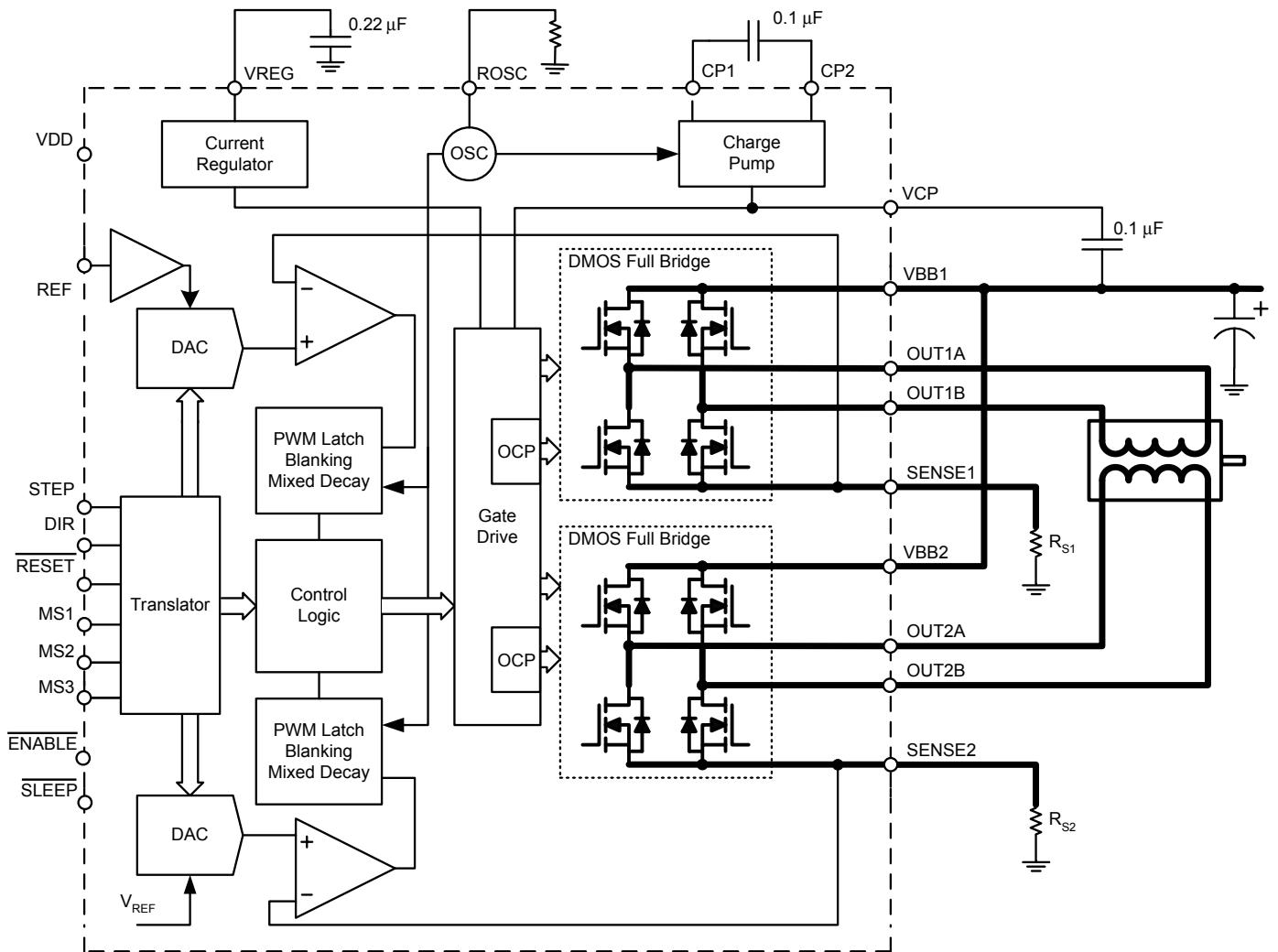
The A4988 is supplied in a surface mount QFN package (ES), 5 mm × 5 mm, with a nominal overall package height of 0.90 mm and an exposed pad for enhanced thermal dissipation. It is lead (Pb) free (suffix -T), with 100% matte tin plated leadframes.

Selection Guide

Part Number	Package	Packing
A4988SETTR-T	28-contact QFN with exposed thermal pad	1500 pieces per 7-in. reel

Absolute Maximum Ratings

Characteristic	Symbol	Notes	Rating	Units
Load Supply Voltage	V_{BB}		35	V
Output Current	I_{OUT}		± 2	A
Logic Input Voltage	V_{IN}		-0.3 to 5.5	V
Logic Supply Voltage	V_{DD}		-0.3 to 5.5	V
Motor Outputs Voltage			-2.0 to 37	V
Sense Voltage	V_{SENSE}		-0.5 to 0.5	V
Reference Voltage	V_{REF}		5.5	V
Operating Ambient Temperature	T_A	Range S	-20 to 85	°C
Maximum Junction	$T_J(max)$		150	°C
Storage Temperature	T_{stg}		-55 to 150	°C

Functional Block Diagram

A4988

DMOS Microstepping Driver with Translator And Overcurrent Protection

ELECTRICAL CHARACTERISTICS¹ at $T_A = 25^\circ\text{C}$, $V_{BB} = 35\text{ V}$ (unless otherwise noted)

Characteristics	Symbol	Test Conditions	Min.	Typ. ²	Max.	Units
Output Drivers						
Load Supply Voltage Range	V_{BB}	Operating	8	—	35	V
Logic Supply Voltage Range	V_{DD}	Operating	3.0	—	5.5	V
Output On Resistance	$R_{DS(ON)}$	Source Driver, $I_{OUT} = -1.5\text{ A}$	—	320	430	$\text{m}\Omega$
		Sink Driver, $I_{OUT} = 1.5\text{ A}$	—	320	430	$\text{m}\Omega$
Body Diode Forward Voltage	V_F	Source Diode, $I_F = -1.5\text{ A}$	—	—	1.2	V
		Sink Diode, $I_F = 1.5\text{ A}$	—	—	1.2	V
Motor Supply Current	I_{BB}	$f_{PWM} < 50\text{ kHz}$	—	—	4	mA
		Operating, outputs disabled	—	—	2	mA
Logic Supply Current	I_{DD}	$f_{PWM} < 50\text{ kHz}$	—	—	8	mA
		Outputs off	—	—	5	mA
Control Logic						
Logic Input Voltage	$V_{IN(1)}$		$V_{DD} \times 0.7$	—	—	V
	$V_{IN(0)}$		—	—	$V_{DD} \times 0.3$	V
Logic Input Current	$I_{IN(1)}$	$V_{IN} = V_{DD} \times 0.7$	-20	<1.0	20	μA
	$I_{IN(0)}$	$V_{IN} = V_{DD} \times 0.3$	-20	<1.0	20	μA
Microstep Select	R_{MS1}	MS1 pin	—	100	—	$\text{k}\Omega$
	R_{MS2}	MS2 pin	—	50	—	$\text{k}\Omega$
	R_{MS3}	MS3 pin	—	100	—	$\text{k}\Omega$
Logic Input Hysteresis	$V_{HYS(IN)}$	As a % of V_{DD}	5	11	19	%
Blank Time	t_{BLANK}		0.7	1	1.3	μs
Fixed Off-Time	t_{OFF}	OSC = VDD or GND	20	30	40	μs
		$R_{OSC} = 25\text{ k}\Omega$	23	30	37	μs
Reference Input Voltage Range	V_{REF}		0	—	4	V
Reference Input Current	I_{REF}		-3	0	3	μA
Current Trip-Level Error ³	err_I	$V_{REF} = 2\text{ V}$, $\%I_{TripMAX} = 38.27\%$	—	—	± 15	%
		$V_{REF} = 2\text{ V}$, $\%I_{TripMAX} = 70.71\%$	—	—	± 5	%
		$V_{REF} = 2\text{ V}$, $\%I_{TripMAX} = 100.00\%$	—	—	± 5	%
Crossover Dead Time	t_{DT}		100	475	800	ns
Protection						
Overcurrent Protection Threshold ⁴	I_{OCPST}		2.1	—	—	A
Thermal Shutdown Temperature	T_{TSD}		—	165	—	$^\circ\text{C}$
Thermal Shutdown Hysteresis	T_{TSDHYS}		—	15	—	$^\circ\text{C}$
VDD Undervoltage Lockout	V_{DDUVLO}	V_{DD} rising	2.7	2.8	2.9	V
VDD Undervoltage Hysteresis	$V_{DDUVLOHYS}$		—	90	—	mV

¹For input and output current specifications, negative current is defined as coming out of (sourcing) the specified device pin.

²Typical data are for initial design estimations only, and assume optimum manufacturing and application conditions. Performance may vary for individual units, within the specified maximum and minimum limits.

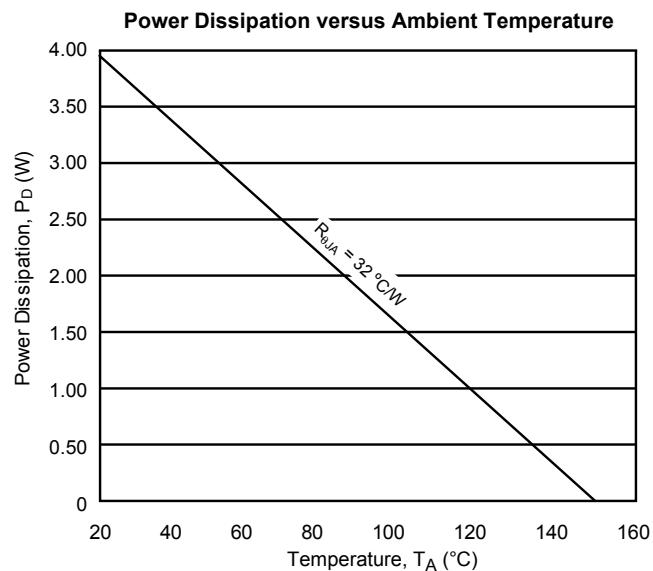
³ $V_{ERR} = [(V_{REF}/8) - V_{SENSE}] / (V_{REF}/8)$.

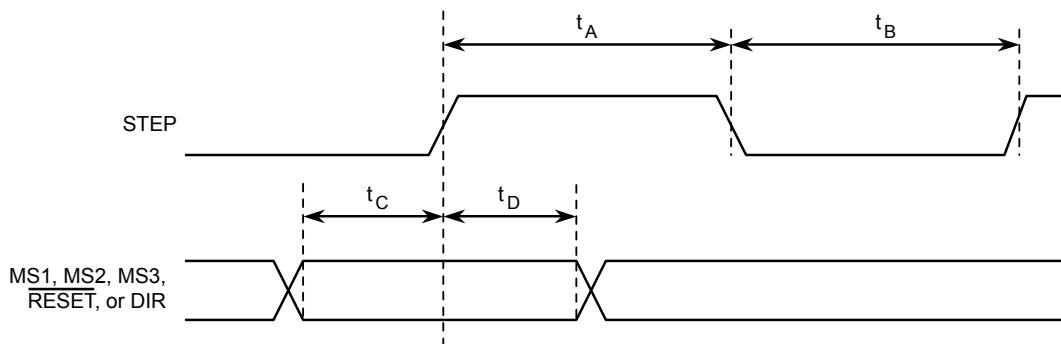
⁴Overcurrent protection (OCP) is tested at $T_A = 25^\circ\text{C}$ in a restricted range and guaranteed by characterization.

THERMAL CHARACTERISTICS

Characteristic	Symbol	Test Conditions*	Value	Units
Package Thermal Resistance	R_{QJA}	Four-layer PCB, based on JEDEC standard	32	$^{\circ}\text{C/W}$

*Additional thermal information available on Allegro Web site.





Time Duration	Symbol	Typ.	Unit
STEP minimum, HIGH pulse width	t_A	1	μs
STEP minimum, LOW pulse width	t_B	1	μs
Setup time, input change to STEP	t_C	200	ns
Hold time, input change to STEP	t_D	200	ns

Figure 1: Logic Interface Timing Diagram

Table 1: Microstepping Resolution Truth Table

MS1	MS2	MS3	Microstep Resolution	Excitation Mode
L	L	L	Full Step	2 Phase
H	L	L	Half Step	1-2 Phase
L	H	L	Quarter Step	W1-2 Phase
H	H	L	Eighth Step	2W1-2 Phase
H	H	H	Sixteenth Step	4W1-2 Phase

Functional Description

Device Operation. The A4988 is a complete microstepping motor driver with a built-in translator for easy operation with minimal control lines. It is designed to operate bipolar stepper motors in full-, half-, quarter-, eighth, and sixteenth-step modes. The currents in each of the two output full-bridges and all of the N-channel DMOS FETs are regulated with fixed off-time PWM (pulse width modulated) control circuitry. At each step, the current for each full-bridge is set by the value of its external current-sense resistor (R_{S1} and R_{S2}), a reference voltage (V_{REF}), and the output voltage of its DAC (which in turn is controlled by the output of the translator).

At power-on or reset, the translator sets the DACs and the phase current polarity to the initial Home state (shown in Figures 9 through 13), and the current regulator to Mixed Decay Mode for both phases. When a step command signal occurs on the STEP input, the translator automatically sequences the DACs to the next level and current polarity. (See Table 2 for the current-level sequence.) The microstep resolution is set by the combined effect of the MSx inputs, as shown in Table 1.

When stepping, if the new output levels of the DACs are lower than their previous output levels, then the decay mode for the active full-bridge is set to Mixed. If the new output levels of the DACs are higher than or equal to their previous levels, then the decay mode for the active full-bridge is set to Slow. This automatic current decay selection improves microstepping performance by reducing the distortion of the current waveform that results from the back EMF of the motor.

Microstep Select (MSx). The microstep resolution is set by the voltage on logic inputs MSx, as shown in Table 1. The MS1 and MS3 pins have a 100 k Ω pull-down resistance, and the MS2 pin has a 50 k Ω pull-down resistance. When changing the step mode the change does not take effect until the next STEP rising edge.

If the step mode is changed without a translator reset, and absolute position must be maintained, it is important to change the step mode at a step position that is common to both step modes in order to avoid missing steps. When the device is powered down, or reset due to TSD or an over current event the translator is set to

the home position which is by default common to all step modes.

Mixed Decay Operation. The bridge operates in Mixed decay mode, at power-on and reset, and during normal running according to the ROSC configuration and the step sequence, as shown in Figures 9 through 13. During Mixed decay, when the trip point is reached, the A4988 initially goes into a fast decay mode for 31.25% of the off-time, t_{OFF} . After that, it switches to Slow decay mode for the remainder of t_{OFF} . A timing diagram for this feature appears on the next page.

Typically, mixed decay is only necessary when the current in the winding is going from a higher value to a lower value as determined by the state of the translator. For most loads automatically-selected mixed decay is convenient because it minimizes ripple when the current is rising and prevents missed steps when the current is falling. For some applications where microstepping at very low speeds is necessary, the lack of back EMF in the winding causes the current to increase in the load quickly, resulting in missed steps. This is shown in Figure 2. By pulling the ROSC pin to ground, mixed decay is set to be active 100% of the time, for both rising and falling currents, and prevents missed steps as shown in Figure 3. If this is not an issue, it is recommended that automatically-selected mixed decay be used, because it will produce reduced ripple currents. Refer to the Fixed Off-Time section for details.

Low Current Microstepping. Intended for applications where the minimum on-time prevents the output current from regulating to the programmed current level at low current steps. To prevent this, the device can be set to operate in Mixed decay mode on both rising and falling portions of the current waveform. This feature is implemented by shorting the ROSC pin to ground. In this state, the off-time is internally set to 30 μ s.

Reset Input (RESET). The RESET input sets the translator to a predefined Home state (shown in Figures 9 through 13), and turns off all of the FET outputs. All STEP inputs are ignored until the RESET input is set to high.

Step Input (STEP). A low-to-high transition on the STEP

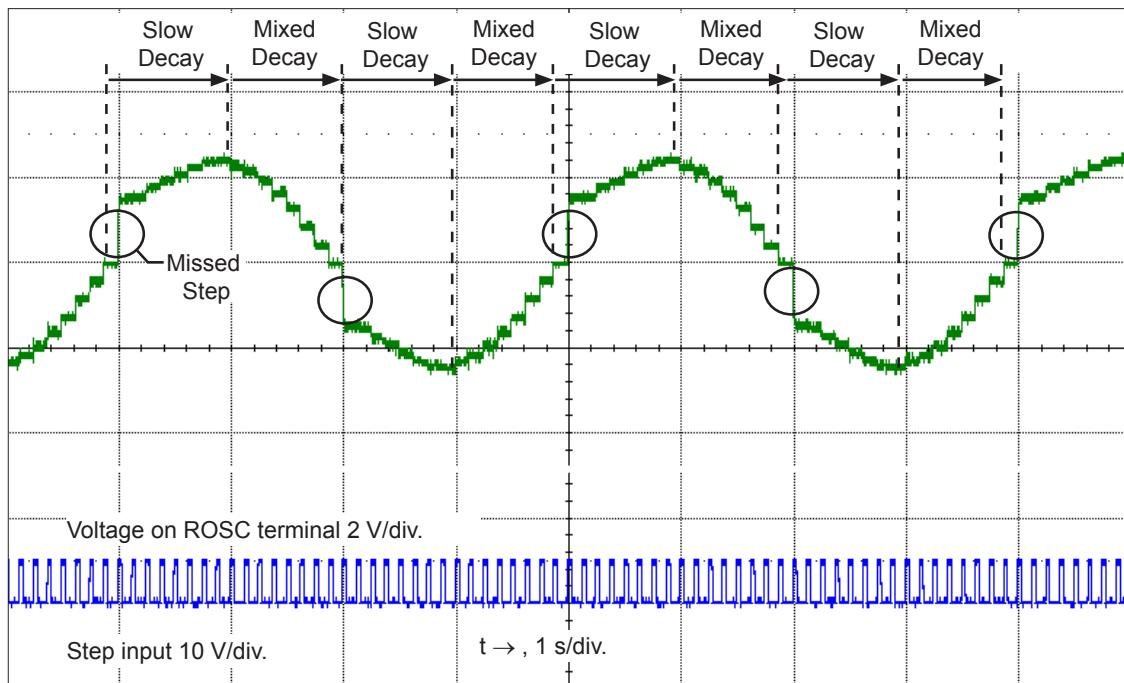


Figure 2: Missed Steps in Low-Speed Microstepping

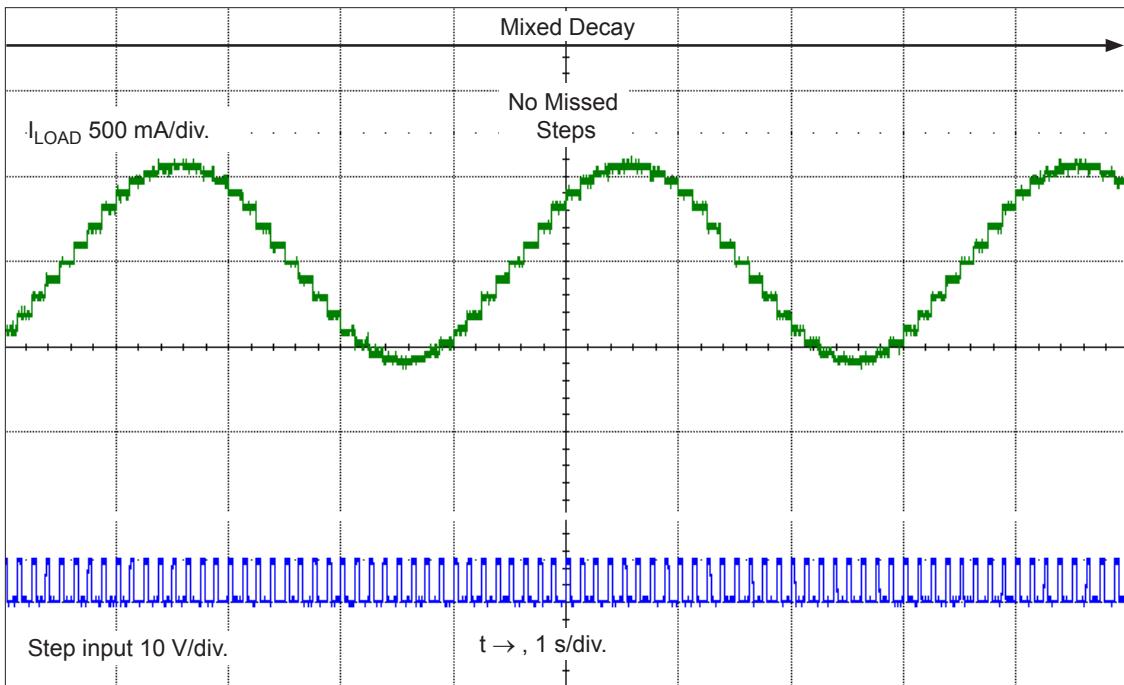


Figure 3: Continuous Stepping Using Automatically-Selected Mixed Stepping (ROSC pin grounded)