UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería Electrónica

Modelo predictivo de detección de fallo en el motor neumático del sistema de enteroscopia Endoworm

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Sistemas Electrónicos

AUTOR/A: Matevosyan, Rafael

Tutor/a: Sánchez Díaz, Carlos

Director/a Experimental: ZAZO MANZANEQUE, ROBERTO

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALENCIA

Dpto. de Ingeniería Electrónica

Modelo predictivo de detección de fallo en el motor neumático del sistema de enteroscopia Endoworm

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Sistemas Electrónicos

Autor/a: Matevosyan, Rafael

Tutor/a: Sánchez Díaz, Carlos

Director/a Experimental: Zazo Manzaneque, Roberto

Curso Académico: 2022/2023

Resumen

Endoworm consiste en un proyecto de investigación con el que se pretende construir un sistema de ayuda a la traslación de un enteroscopio por el interior del intestino delgado. Tal como indica el nombre del proyecto, el movimiento que sigue el equipo se asemeja al de un gusano. Para conseguir el mismo desplazamiento, se ha dotado al enteroscopio de un total de cuatro cavidades hinchables y un sistema electrónico que se encarga de controlarlos.

Dentro de este proyecto, el presente Trabajo Final de Máster se centra en implementar una interfaz gráfica y dos modelos predictivos de clasificación. La interfaz gráfica se emplea para que el usuario se pueda comunicar con el equipo y los modelos predictivos permiten discernir si los dos tipos de cavidades presentan una fuga de aire o no. Existen un total de cuatro señales, pero se pueden agrupar en dos grupos: expansión axial y expansión radial. Para obtener dichos modelos será necesario realizar el procesado de las señales de presión de ambos tipos de cavidades, para obtener una serie de características. A partir de estas características, se entrenarán los modelos de Machine Learning, y una vez que estén validados se incorporarán al software del dispositivo de control para poder detectar fugas.

Resum

Endoworm consistix en un projecte d'investigació amb el qual es pretén construir un sistema d'ajuda a la translació d'un enteroscopi per l'interior de l'intestí prim. Tal com indica el nom del projecte, el moviment que segueix l'equip s'assembla al d'un cuc. Per a aconseguir el mateix desplaçament, s'ha dotat l'enteroscopi d'un total de quatre cavitats inflables i un sistema electrònic que s'encarrega de controlar-les.

Dins d'aquest projecte, el present Treball Final de Màster se centra en implementar una interfície gràfica i dos models predictius de classificació. La interfície gràfica s'emplea perquè l'usuari puga comunicar-se amb l'equip i els models predictius permeten discernir si els dos tipus de cavitats presenten una fuga d'aire o no. Hi ha un total de quatre senyals, però es poden agrupar en dos grups: expansió axial i expansió radial. Per a obtindre aquests models serà necessari realitzar el processament de les senyals de pressió d'ambdós tipus de cavitats, per a obtindre una sèrie de característiques. A partir d'aquestes característiques, s'entrenaran els models de Machine Learning, i una vegada estiguen validats s'incorporaran al codi del dispositiu de control per a poder detectar fugues.

Abstract

Endoworm is a research project aimed at constructing a system to assist in the translational movement of an enteroscope within the small intestine. As the project's name suggests, the motion the equipment follows resembles that of a worm. To achieve this movement, the enteroscope has been equipped with a total of four inflatable cavities and an electronic system responsible for their control.

Within this project, the present Master's Final Project focuses on implementing a graphical interface and two predictive classification models. The graphical interface is used to allow the user to communicate with the equipment, and the predictive models help discern whether the two types of cavities have an air leak or not. There are a total of four signals, but they can be grouped into two categories: axial expansion and radial expansion. To obtain these models, it will be necessary to process the pressure signals from both types of cavities to obtain a set of features. Based on these features, the Machine Learning models will be trained, and once validated, they will be incorporated into the control device's software to detect leaks.

Índice

Contenido del documento

- Memoria
- Presupuesto
- Pliego de condiciones

ÍNIDCE DE LA MEMORIA

ÍNDICE DEL PRESUPUESTO

ÍNDICE DEL PLIEGO DE CONDICIONES

Índice de figuras

Índice de tablas

UNIVERSITAT POLITÈCNICA DE VALENCIA

Dpto. de Ingeniería Electrónica

Modelo predictivo de detección de fallo en el motor neumático del sistema de enteroscopia Endoworm

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Sistemas Electrónicos

MEMORIA

Autor/a: Matevosyan, Rafael

Tutor/a: Sánchez Díaz, Carlos

Director/a Experimental: Zazo Manzaneque, Roberto

Curso Académico: 2022/2023

Índice de la memoria

ÍNIDCE DE LA MEMORIA

Capítulo 1. Introducción

El tracto gastrointestinal presenta una longitud de aproximadamente 9,5 metros, está formado por el esófago, estomago e intestinos y se encuentra dividido en dos partes el tracto gastrointestinal superior e inferior. Incluye todas las estructuras que se encuentran entre la boca y el ano, formando un pasadizo que incluye los principales órganos de digestión, que son el estómago, el intestino delgado y grueso. Para el diagnóstico de patologías del intestino delgado en pacientes, se emplean técnicas enteroscópicas. Algunas de las patologías más frecuentes son hemorragias intestinales, extirpación de pólipos, toma de biopsias…. La principal ventaja de este tipo de técnicas es que son mínimamente invasivas y posibilitan el diagnóstico, así como la aplicación de tratamientos.

Los problemas de exploración del intestino delgado fueron parcialmente solucionados mediante la capsula endoscópica en el año 2000 [1]. Su empleo está ampliamente extendido para el diagnóstico de patologías, ya que es una herramienta potente y tiene la ventaja de ser inocua para el paciente. Sin embargo, presenta el inconveniente de que carece de herramientas para la realización de intervenciones quirúrgicas. En tal caso, se suele emplear el enteroscopio de doble balón, que apareció en el 2001, este presenta varios utensilios quirúrgicos ideales para realizar todo tipo de intervenciones en el intestino delgado.

A lo largo de los años se han llevado a cabo varias investigaciones, tanto para conseguir una capsula endoscópica que aparte de realizar exploraciones completas y de diagnosticar enfermedades que fuera capaz de tratar enfermedades como de otros métodos para mejorar la técnica de doble balón. Respecto a la capsula endoscópica existen modelos capaces de realizar una exploración completa y de diagnosticar enfermedades, pero en lo que implica tratar enfermedades solo existen prototipos que no han ido más allá de la etapa de ensayo, [1] sin embargo, han aparecido alternativas a la técnica de doble balón como el mono balón o el espiral, disponibles en el mercado y ampliamente extendido su uso.

Varios prototipos han incluido sistemas robóticos de locomoción, algunos inspirados en la naturaleza y otros por sistemas mecánicos. De hecho, en 1995 se publicaron en Grundfest et al los resultados preliminares de un robot endoscopio basado en actuadores neumáticos con capacidad de iluminar y obtención de imágenes [2]. Desde entonces, la nueva generación de robots endoscopios han ido mejorando, obteniendo mejores resultados.

El prototipo que se describe en este trabajo es un sistema semi autónomo que combina la robótica con los balones del sistema estereoscópico [3]. El sistema, aunque puede ser autopropulsado, no es capaz de mover por sí solo un endoscopio convencional a lo largo del intestino delgado. Esto se debe a que necesita la asistencia de un especialista que vaya replegando o retirando, depende de si el sistema está avanzando o retrocediendo, el intestino que vaya moviendo el sistema. Esta es la diferencia principal que distingue este proyecto del resto de sistemas que se han ido desarrollando, los cuales son completamente manuales, es decir, que necesitan, para ejercer el movimiento de replegado del intestino, la intervención de un especialista mediante el deslizamiento de un sobretubo por el endoscopio para lograrlo.

1.1 Justificación

Una endoscopia es una técnica de diagnóstico que permite explorar dentro de una cavidad, conducto u órgano hueco, este tipo de operaciones son realizadas mediante un endoscopio, el cual se introduce a través un orificio natural o a mediante una incisión quirúrgica.

Un endoscopio está formado, tal como se observa en la *Figura 1*, por una sonda flexible que presenta un foco de luz para iluminar la cavidad, una cámara para obtener imágenes del interior, un sistema capaz de controlar la punta flexible y una serie de canales que sirven como pasaje de

aire, agua e incluso material quirúrgico para tomar muestras y analizarlas, para quitar pólipos, para cauterizar lesiones vasculares que puedan sangrar. Es por este motivo que este tipo de herramientas son ampliamente empleados en la medicina como técnicas de prevención, diagnóstico, así como tratamiento de diversas patologías e intervenciones quirúrgicas.

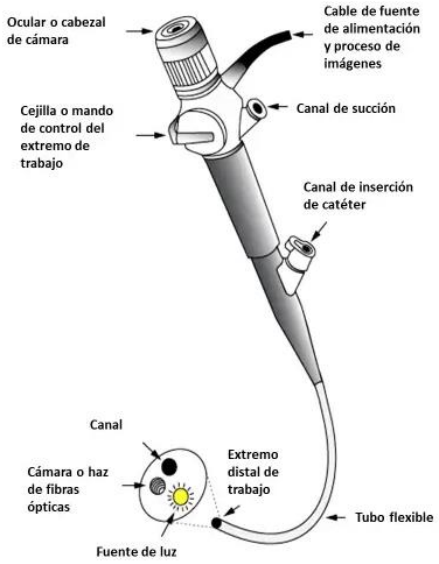


Figura 1. Partes de un endoscopio semiflexible. [4].

La endoscopia en el aparato digestivo se puede dividir en cuatro grupos:

- **Gastroscopia:** empleado en la visualización del esófago, el estómago y el duodeno.
- **Colonoscopia:** consiste en una exploración del intestino grueso y la parte final del intestino delgado (íleon terminal).
- **Enteroscopia:** enfocado en la exploración del intestino delgado.
- **Colangiopancreatografia retrógrada endoscópica:** empleado para el diagnóstico del páncreas y la vía biliar.

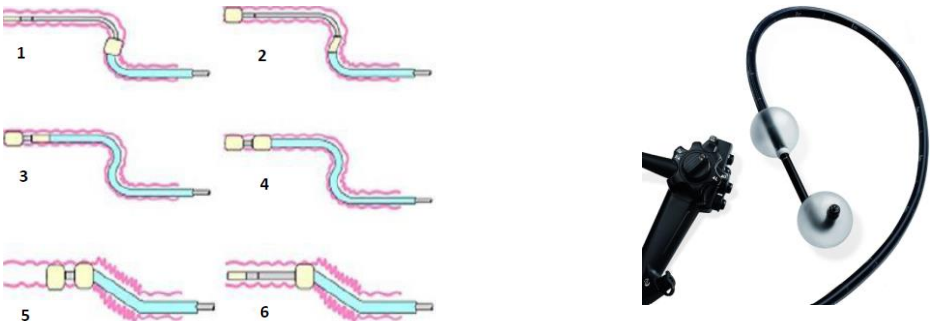
Este proyecto se centra en el endoscopio empleado para la exploración del intestino delgado, denominado enteroscopio.

Actualmente en sanidad existen principalmente tres tipos de técnicas para la enteroscopia [5], las cuales son: mono-balón (SBE - Single Balloon Enteroscopy), doble-balón (DBE - Double Balloon Enteroscopy) y de espiral (SE - Spiral Enteroscopy). Se va a explicar cada una de estas con más detalle.

1.1.1 Enteroscopia doble-balón (DBE)

Fue desarrollado en 2001 por Hironori Yamamoto para la evaluación completa desde el yeyuno hasta el íleon. El DBE emplea un endoscopio acoplado, un sobretubo con dos balones de látex colocados en los extremos distales de cada componente. Ambos balones se emplean para fijar el endoscopio al intestino delgado, de modo que pueda avanzar y replegar el intestino. Actualmente existen tres tipos de DBE, el más utilizado presenta un diámetro de 9.4mm y una longitud de 200cm (EN-450T5), pero existen otros con un diámetro más pequeño (EN-450P5/20) y otros más cortos (EC-450BI5). Los de diámetro pequeño puede emplearse en pediatría y para diagnóstico en adultos. Los de longitud corta se emplean en ileocolonoscopias difíciles, para el CPRE en el postoperatorio para diagnosticar complicaciones en la anatomía del intestino

delgado[6]. En la *Figura 2* se puede observar un enteroscopio de doble balón y un esquema del mecanismo de avance.



(a) *(b)*

Figura 2. Pasos del mecanismo de avance de un enteroscopio de doble balón(a). Imagen de un enteroscopio de doble balón(b). [7].

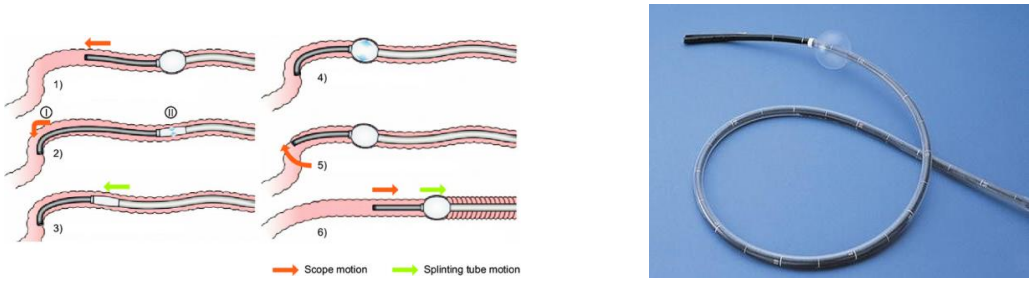
El mecanismo de avance es el siguiente:

1. Sobretubo fijado al intestino y teniendo el balón de sobretubo inflado, se inserta el endoscopio para que pueda avanzar la máxima distancia posible a lo largo del intestino.
2. El balón de la punta del endoscopio se infla, para que se quede fija, y luego se desinfla el balón del sobretubo.
3. El sobretubo se desplaza hacia delante hasta aproximarse lo máximo posible a la posición del balón del endoscopio.
4. Se infla el balón del sobretubo, dejando como consecuencia ambos fijados al intestino.
5. Con el objetivo de replegar el intestino, se mueve hacia detrás del sobretubo y del enteroscopio.
6. El balón de la punta se deshincha.

Este proceso se repite de forma cíclica para que el endoscopio avance a lo largo del intestino delgado.

1.1.2 Enteroscopia mono-balon (SBE)

Este tipo de técnica fue desarrollada por Olympus en 2007. A diferencia del DBE presenta un único balón colocado al final del sobretubo. Al igual que el DBE sigue la técnica de push-pull, es decir emplear el balón para hacer avanzar el endoscopio, push, y también el plegado y retirada del intestino delgado sobre el endoscopio, pull. En la *Figura 3* se puede observar el mecanismo de avance de un enteroscopio mono-balón.



(a) *(b)*

Figura 3. Pasos del mecanismo de avance de un enteroscopio mono balón (a [8]*). Imagen de un enteroscopio de mono balón(b)* [9]

El mecanismo de avance es el siguiente:

1. Teniendo el sobretubo fijado a las paredes del intestino y con el balón inflado, el endoscopio avanza lo máximo posible a lo largo del intestino delgado.
2. Se va girando la punta móvil del endoscopio, la cual tiene forma de gancho, hasta que el endoscopio se queda fijado al intestino y se desinfla el balón del sobretubo para desfijarlo.
3. Teniendo el endoscopio fijado, se hace avanzar el sobretubo, mediante un giro de la punta móvil,
4. El balón del sobretubo se infla para que se quede fija al intestino.
5. El endoscopio deja de estar sujeto al intestino poniendo la punta móvil recta.
6. Con el propósito de replegar el intestino, se mueve hacia detrás del sobretubo y del endoscopio.

Con la finalidad de conseguir un movimiento continuo del endoscopio estos seis pasos se deben de repetir de forma cíclica.

1.1.3 Enterosocopia en espiral (ES)

La enteroscopia en espiral fue introducido por Akerman y Cantero en 2006. En la ES se acopla a un endoscopio un sobretubo que presenta un relieve en forma de hélice en su extremo distal de tal manera que, mediante un mecanismo rotario en sentido horario, imitando el movimiento de un sacacorchos, se transforma dicha fuerza de rotación en una fuerza lineal que ayuda a plegar el intestino delgado y así avanzar a través de él [10]. En la *Figura 4* se muestra el mecanismo de avance de un enteroscopio en espiral.

(a) *(b)*

Figura 4. Mecanismo de avance de un enteroscopio en espiral(a). Imagen de un enteroscopio en espiral (b).[10] .

El mecanismo de avance presenta los siguientes pasos:

1. El endoscopio es introducido por la boca con el sobretubo montado.
2. Se hace avanzar el enteroscopio mediante empujes y giros hasta que se pasa el píloro y se llega al intestino delgado.
3. Se realiza el giro del sobretubo en el sentido de las agujas del reloj. Al llegar al punto en que la rotación del sobretubo ya no es efectiva, se desbloquea el enteroscopio del sobretubo, y se hace avanzar hasta que se pueda.
4. Una vez que se llega al tope de avance del enteroscopio, se sujeta a la pared del intestino delgado, al mismo tiempo que se retira el enteroscopio, rotando el sobretubo suavemente en el sentido antihorario, de tal modo que se despliega el intestino delgado.

Para la retirada del ES se deben de realizar movimientos antihorarios, lentamente, permitiendo poco a poco la liberación del intestino delgado. Esta técnica es más sencilla y rápida de llevarse a cabo que las dos anteriores.

1.1.4 Comparación y justificación del proyecto

Una vez descritas cada una de las técnicas que existen, se va a realizar una comparativa entre cada uno de estas para determinar las ventajas y desventajas que existen y de este modo detectar las necesidades que justifiquen el desarrollo del proyecto Endoworm. Las variables que se han estudiado han sido: enteroscopia completa, profundidad de inserción, duración, curva de aprendizaje y complicaciones [11].

Author	Design	Patient no.	Depth of insertion (cm), antegrade vs retrograde		Procedure time (min), antegrade vs retrograde		Overall adverse event rate, %
DBE vs SBE							
Efthymiou et al (2012)	RCT	66 vs 53	234 vs 204	75 vs 72	60 vs 60		1.5 vs 1.8
Domagk et al (2011)	RCT	65 vs 65	253 vs 258	107 vs 118	105 vs 96		0
May et al (2010)	RCT	50 vs 50	–	–	67 vs 54	62 vs 60	4 vs 8
Lenz et al (2013)	Retrospective	1052 vs 515	245 vs 218	–	50 vs 40	55 vs 46	0
DBE vs spiral							
Rahmi et al (2013)	Prospective	191 vs 50	200 vs 220	–	60 vs 55	–	24 vs 18
Messer et al (2013)	RCT	13 vs 13	346 vs 268	209 vs 78	60 vs 43	76 vs 52	23 vs 23 (1 perforation in retrograde spiral)
May et al (2011)	RCT	10 vs 10	310 vs 250	–	65 vs 43	–	–
Frieling et al (2010)	Prospective	17 vs 18	260 vs 250	–	42 vs 47	–	0
SBE vs spiral							
Khashab et al (2010)	Retrospective	52 vs 53	222 vs 301	–	53 vs 47	–	3.8 vs 1.9 (1 perforation in single-balloon)

Tabla 1.Comparativa entre SBE, DBE y ES, datos relacionados con el procedimiento[11]

Author	Design	No. of procedures	Diagnostic yield, %	Therapeutic yield, %
DBE vs SBE				
Efthymiou et al (2012)	RCT	66 vs 53	53 vs 57	26 vs 32
Domagk et al (2011)	RCT	65 vs 65	43 vs 37	9 vs 5
Takano et al (2011)	RCT	20 vs 18	50 vs 61	35 vs 27.8
May et al (2010)	RCT	50 vs 50	52 vs 42	72 vs 48
Lenz et al (2013)	Retrospective	1052 vs 515	48.2 vs 61.7	–
DBE vs spiral				
Messer et al (2013)	RCT	13 vs 13	46 vs 69	92
Frieling et al (2010)	Prospective	17 vs 18	47.1 vs 33.4	–
SBE vs spiral				
Khashab et al (2010)	Retrospective	52 vs 53	59.6 vs 43.4	33 vs 15

Tabla 2. Comparativa entre SBE, DBE y ES, resultados [11]

1. **Profundidad de inserción**

   Este parámetro determina la distancia máxima que el enteroscopio es capaz de avanzar a lo largo del intestino delgado, lo cual es una característica importante, ya que a mayor distancia mayor cantidad de intestino explorado. Sin embargo, presenta una limitación a la hora de determinar cuánto ha llegado a avanzar el enteroscopio, cuestión no totalmente resuelta con la medición secuencial de longitud usada por el grupo Wiesbaden [12] ni con la medición de la inserción del sobretubo según Li et al, [13], 5cm de avance de este equivale a 40 cm del enteroscopio.

Tal como se observa en la *Tabla 1*, se han realizado un total de 3 estudios donde se comparan la profundidad de inserción del DBE y SBE por las dos vías, oral y anal. En el estudio de *Efthymiou et al* la distancia media alcanzada por vía oral fue de 234 cm con DBE frente a 204 cm y por vía anal de 75 vs 72 cm, no siendo estas diferencias estadísticamente significativas. En el estudio de *Dogmak et al*, la distancia avanzada, por vía oral, en DBE frente a SBE fue de 253 y 258 cm y por vía anal 107 y 118 cm, sin observarse tampoco diferencias significativas. En el caso del estudio de *Lenz et al* hubo un mayor número de casos, pero igualmente mostró que no hay mucha diferencia entre DBE y SBE,

En la comparación entre DBE y el SE ha mostrado unos resultados incongruentes. Se han realizado un total de 4 estudios donde se ha analizado la profundidad de inserción por vía oral y en dos de ellos, *Messer et al* y *May et al* se ha mostrado que el DBE puede alcanzar una mayor distancia que el SBE, sin embargo, en estudios grandes como el de *Rahmi et* al, se puede observar que esta diferencia se reduce considerablemente. Solo en el estudio de *Messer et al* se ha llegado a comparar ambos por vía anal y como resultado se ha obtenido que mediante el DBE se alcanza una mayor distancia que el SE.

Un único estudio ha realizado la comparación entre SBE y SE por vía oral, en este se ha observado que el SE alcanza una mayor distancia que el SBE.

Como conclusión de la *Tabla 1* se puede indicar que el DBE es el que presenta un alcance mayor, sin embargo, no se aleja mucho del SE y DBE. A pesar de que en el estudio de *Khashab et al* muestre que el SE tiene un mayor alcance que el SBE, en el resto de estudios donde se comparan el DBE vs SBE y DBE vs SE, muestran que entre ellos la profundidad de inserción es similar.

2. Enteroscopia completa

El ratio de enteroscopias completas indica el número de casos en los que se ha conseguido una visualización completa del intestino delgado, esto desde la vía oral hasta la anal, y el número de casos que se ha intentado.

Un estudio realizado por *May et al*, donde se compararon, con un total de 50 pacientes, el DBE y el SBE, indico que en el caso de DBE se conseguían un mayor caso de completos para el DBE que el SBE (66% contra 22 %%). Más adelante se llevó a cabo otro estudio donde se compararon las mismas técnicas y dio un resultado de 57% para el DBE y un 0 % para el SBE, este estudio fue realizado por *Takano et al*. Sin embargo, fue criticado porque los autores tenían mucha más experiencia con DBE que con SBE, de modo que habían realizado 248 DBE frente a solo 10 SBE antes de empezar el estudio.

Otro de los estudios fue el de *Messer et al*, en este se compararon el DBE y el SE en un total de 26 pacientes. El DBE alcanzo un total de 92% mientras que el SE un 8%.

En general, el impacto clínico de este parámetro se ha mantenido en duda porque por que los diagnósticos y las operaciones pueden llegar a completarse sin la necesidad de realizar una enteroscopia completa. Además, en muchos casos no está claro si una enteroscopia completa fue el objetivo principal del estudio. A pesar de ello, el número de casos completos es mayor en el DBE. Sin embargo, el uso de este parámetro para determinar la eficiencia de cada una de las técnicas no está del todo claro.

3. Duración

La duración de una enteroscopia puede estar afectada por varios factores, incluyendo características propias del paciente, como puede ser la obesidad, historial clínico, número de afecciones o patologías presentes en el tracto digestivo, etc. Pocos estudios han

comparado directamente el tiempo de asistencia entre cada una de las técnicas. En un estudio realizado por *Khashab et al*, se comparó el SBE y el SE, los resultados no mostraron mucha diferencia, 53 min y 47 min respectivamente. Datos publicados anteriormente sugerían que los tiempos medios para el DBE, SBE y SE por vía oral era de 70, 60 y 40 min y en caso de vía anal 65, 69 y 46 min respectivamente. Por lo tanto, el SE parece ser aquel que presenta unos tiempos más cortos.

4. Rentabilidad diagnóstica y terapéutica

Indica el porcentaje de casos en que se confirma el diagnóstico de sospecha por medio de la enteroscopia o bien que se observa una patología que influye en el manejo del paciente. Se trata de un parámetro que indica la relevancia clínica de la técnica. La rentabilidad terapéutica hace referencia al número de intervenciones terapéuticas y es un parámetro importante a la hora de comparar el impacto clínico entre los distintos sistemas de enteroscopia.

En cuatro estudios comparativos entre el SBE y DBE, la rentabilidad diagnóstica para ambos fue del 40% al 60% respectivamente y la rentabilidad terapéutica sobre un 30%. Solo en un estudio se mostró una mayor rentabilidad terapéutica de la DBE, del 72% comparado con el 48% del SBE. La rentabilidad diagnóstica de la SBE y el SE fue similar en un estudio comparativo, del 59.6% y 43.4%, respectivamente. Los resultados de todos estos estudios están en la *Tabla 2*. Como resultado se puede indicar que la rentabilidad diagnóstica y terapéutica es similar entre los distintos sistemas, a pesar de que algunos estudios muestran un mayor porcentaje en el DBE.

5. Curva de aprendizaje

No hay estudios que directamente comparen la curva de aprendizaje entre las tres técnicas. Sin embargo, en el caso del DBE y SBE se han reportado mejoras en los tiempos y cantidad de intestino explorada después de 10 a 15 procedimientos de enteroscopia asistidos [14], [15]. Se presenta una curva de aprendizaje más corta, del orden de 5 entrenamientos completos para realizar una correcta operación [16].

6. Complicaciones

Tal como se observa en la *Tabla 1*, el número de complicaciones que pueden llegar a producirse durante una enteroscopia, es similar en las tres técnicas, siendo el SBE el que ha presentado un número ligeramente superior de casos donde ha surgido algún problema. Este parámetro, depende de varios factores, como puede ser del enteroscopista, del paciente, del equipo o de forma externa, por lo que no es del todo dependiente del tipo de técnica empleada.

Conclusión

Una vez analizados cada una de las técnicas se ha podido observar cuatro necesidades que justifiquen el desarrollo de una nueva técnica:

1. Disminuir la duración de la exploración a tiempos similares a los de SE manteniendo un ratio de exploraciones completas similar al de DBE.
2. Disminuir la curva de aprendizaje necesaria por parte del personal sanitario para operar el enteroscopio.
3. Mantener una rentabilidad diagnóstica y terapéutica similar al DBE.
4. Reducir el número de complicaciones que puedan aparecer durante la exploración.

1.2 Antecedentes

El sistema Endoworm es un dispositivo semi autónomo que apoya al especialista a operar con el endoscopio dentro del intestino delgado. Este proyecto comenzó en el año 2005 y desde entonces se han obtenido varias versiones, actualmente se encuentra en la versión v.4.0. Su objetivo consiste en replegar el intestino sobre el endoscopio para que este pueda avanzar y explorar así el tracto digestivo. Está formado por una sonda enteroscopica comercial y un dispositivo de control.

1.2.1 Versión anterior

En la versión anterior del sistema, v.3.0, la sonda enteroscopica, estaba conformada por dos balones y un fuelle que eran inflados y desinflados mediante un circuito neumático que se encontraba dentro del dispositivo de control, siendo gobernado por un microcontrolador. Los balones de la sonda enteroscópica se encargaban de fijar el endoscopio al intestino delgado mientras que el fuelle se encargaba de hacer avanzar el endoscopio cuando los balones se encontraban deshinchados [17].

El circuito neumático del dispositivo de control estaba formado por una bomba de presión que almacenaba aire comprimido en un calderín y una bomba aspiradora que mantenía una presión de aire negativa, con respecto la atmosférica, en otro calderín. El software del microcontrolador, encargado de gobernar el dispositivo de control, había sido diseñado para conseguir una coordinación de hinchado y deshinchado de las cavidades [17].

La *Figura 5* muestra un diagrama de bloques del hardware empleado en la versión 3.0 del sistema Endoworm.

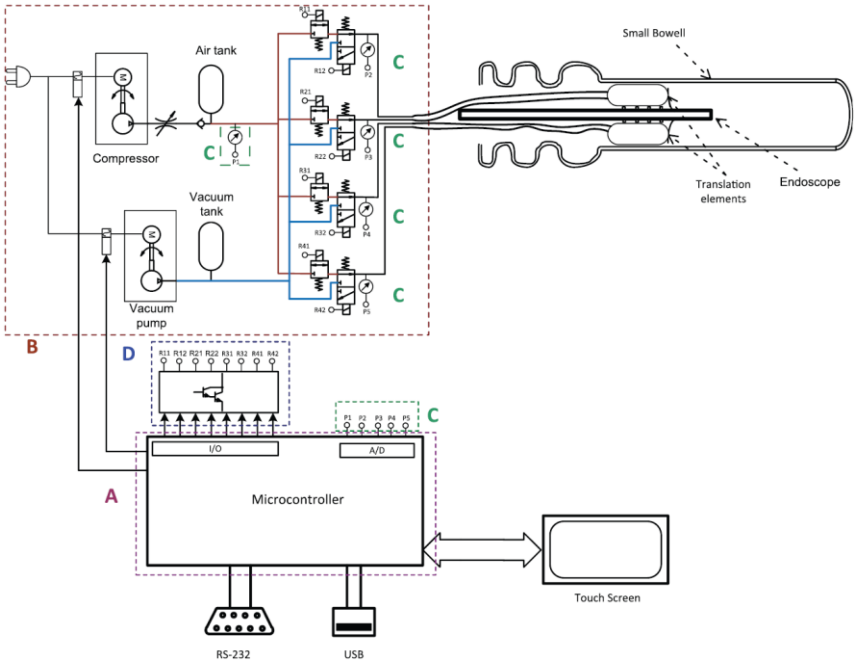


Figura 5. Diagrama de bloques del sistema de control hardware.[3]

Tal como se observa en la *Figura 5* existen tres grupos principales en los que se puede separar el sistema, los cuales son:

1. **Circuito neumático:** permite el movimiento coordinado de las distintas cavidades de la sonda enteroscópica. Este a su vez está formado por los siguientes elementos:
   - **Calderines:** para almacenar aire por encima y por debajo de los niveles de presión ambiental. Su objetivo es generar un gradiente de presiones de modo que se origine un caudal que permita inflar y desinflar las cavidades de expansión axial.
   - **Bombas de presión:** su objetivo consiste en la de generar presiones superiores e inferiores a la presión ambiente para poder hinchar y deshinchar cada una de las cavidades.
   - **Electroválvulas:** necesario para regular la entrada de aire en las cavidades.
   - **Conductos:** para poder interconectar los distintos sistemas y poder cerrar el circuito neumático.

2. **Sonda enteroscopica**: está formado por una sonda flexible, que permite iluminar y captar imágenes del interior del intestino y una serie de herramientas con las que se puede realizar varios tipos de intervenciones. Sobre la sonda se colocan las distintas cavidades, que son los encargados de fijar y mover el enteroscopio. Cada uno de ellos presenta un nombre especifico que indica cuál es su función:
   - CEA: cavidad de expansión axial.
   - CER móvil: cavidad de expansión radial móvil.
   - CER fijo: cavidad de expansión radial fijo.

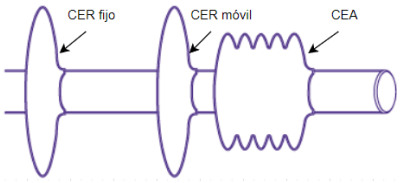


Figura 6. Representación de cada una de las cavidades [18].

3. **Sistema electrónico**: su función consiste en la de generar las señales necesarias para poder abrir y cerrar cada una de las cavidades, de modo que se genere el movimiento del endoscopio a lo largo del intestino delgado. El conjunto de elementos que lo forman se puede observar en la *Figura 7*.

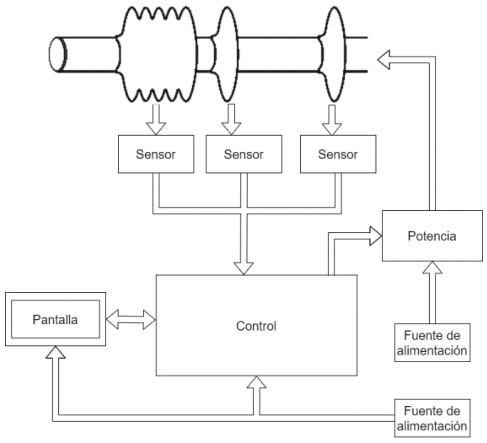


Figura 7. Estructura del sistema electrónico del equipo Endoworm 3.0 [Elaboración propia]

- **Control:** su función consiste en generar los impulsos necesarios para la activación de cada una de las cavidades.
- **Potencia:** se encarga de generar la corriente necesaria para poder activar cada una de las electroválvulas, ya que con la etapa de control no se generaban los valores corriente necesarios.
- **Sensores de presión:** encargados de medir la presión diferencial en cada una de las cavidades.
- **Pantalla inteligente:** permite la comunicación entre usuario y la placa de control.
- **Fuentes de alimentación:** encargadas de generar los valores de tensión y corriente necesarios para poder alimentar cada una de las partes que forma el sistema electrónico.

A parte de estos dos sistemas existan otra serie de elementos que forman parte del equipo como puede ser los tornillos, la propia carcasa, los cables, ventiladores, filtros para evitar las interferencias electromagnéticas generadas por la red eléctrica sobre el dispositivo de control y viceversa.

1.2.2 Versión actual

Respecto al modelo anteriormente descrito se han realizado una serie de mejoras, que han dado lugar a la versión 4.0 del sistema Endoworm. Los cambios más significativos que se han realizado en esta nueva versión han sido los siguientes.

El empleo de un microcontrolador de 32 bits en vez de 8 bits, el microcontrolador que se estaba empleando era el *PIC18F4550* de la familia Microchip. Ahora el que se va a emplear pertenece a la familiar de *ST Electrónics*, el modelo es *STM32F429ZTIx*, en el Anexo A4 se puede ver las características de este microcontrolador. El motivo por el que se decidió cambiar de microcontrolador fueron las limitaciones de cómputo del PIC, que eran insuficientes para la implementación del sistema de detección de fallos de la sonda enteroscópica, el cual se describirá en detalle los capítulos 4 y 5. Mientras que el STM, al disponer de funcionalidades DSP y mayor poder de computación, sí que permite implementar el sistema de detección de fallos.

También se cambió el tipo de sensores de presión, ahora se emplean sensores capaces de medir la presión absoluta (SSCDANN100PAAA5 de Honeywell) en lugar de la relativa (26PCDF A2G de Honeywell). Este cambio permite obtener mucha más información de las cavidades que antes, ya que ahora permite registrar presiones por debajo de la presión atmosférica. Con el modelo de sensor anterior y la electrónica empleada no se podían captar dichas presiones. Con esta nueva información de presiones se cree que se puede mejorar la capacidad de detección de fugas de aire en la sonda enteroscópica.

Otro cambio relevante es relativo a la utilización de una interfaz gráfica con el usuario (GUI), que se ha cambiado una del ámbito industrial de la marca *Proface* a una de la marca *Emerging Display Technologies CORP,* en el Anexo A3 se encuentran la hoja de datos del microcontrolador que lleva incorporado. La cual presenta un diseño mejor acabado, es más fino y lo más importante de todo, la facilidad que presenta a la hora de programar la pantalla ya que dispone de un microcontrolador *ARM Cortex-M7*, por lo que existe una gran cantidad de información y librerías muy útiles. Además de posibilitar una mayor flexibilidad a la hora de realizar el diseño y la programación funcional de la interfaz gráfica.

Respecto a la sonda entersocópica se ha añadido una cavidad adicional, por lo que ahora existen un total de 4 cavidades. Estas son:

1. **DAEC:** globo de expansión axial distal (distal axial expansion cavity).
2. **PAEC:** globo de expansión axial proximal (proximal axial expansion cavity).
3. **MREC**: globo de expansión radial móvil (mobile radial expansion cavity).

4. **FREC:** globo de expansión radial fijo (fix radial expansion cavity).

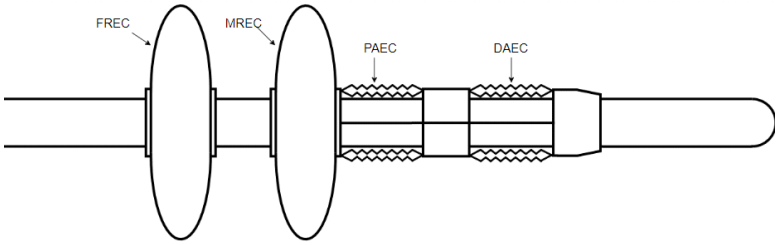


Figura 8. Estructura actual de las cavidades de la versión 4.0. [Elaboración propia]

El introducir una cavidad adicional ha supuesto hacer un cambio integro en el resto de componentes que conforman el dispositivo de control, que van desde el aumento de la capacidad de los tanques de aire, la introducción de electroválvulas adicionales, la modificación íntegra del diseño electrónico del dispositivo, entre otras. Además, este cambio implica una modificación de la secuencia de acciones de hinchado y deshinchado de cada uno de los globos para que la sonda enteroscopica pueda avanzar o retroceder a lo largo del intestino delgado. Dichas acciones se van a detallar a continuación.

El avance del sistema se puede observar en la *Figura 9*, la cual consiste en:

1. El sistema parte en una posición de reposo, en el cual todas las cavidades se encuentran deshinchadas.
2. Se hincha el DAEC, con el objetivo de mover hacia adelante el MREC.
3. Se infla la cavidad MREC, esta posición da comienzo al bucle.
4. Se deshincha el DAEC y se hincha el PAEC (de forma antagonista), de modo que se juntan las cavidades de expansión radial (FREC y MREC), de esta forma se ha arrastrado parte del intestino hacia atrás.
5. Se hincha el FREC, de modo que este se ancla a las paredes del intestino delgado, con todo el intestino que se había replegado mediante la acción 4.
6. Se deshincha el MREC.
7. El PAEC se deshincha y el DAEC se hincha con la finalidad de desplazar la cavidad móvil.
8. En esta posición se vuelve a hinchar el MREC, de modo que la cavidad MREC está listo para volver a desplazar parte del intestino hacia atrás. Mediante esta acción de fijar y desplazar hacia atrás se consigue que la sonda vaya avanzando hacia delante.

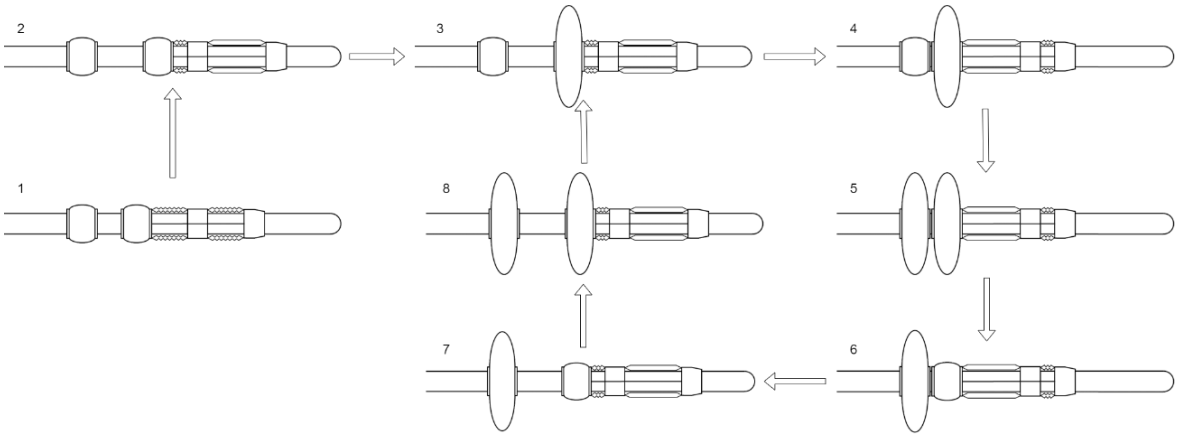


Figura 9. Secuencia de avance. [Elaboración propia]

La secuencia de retroceso se puede observar en la *Figura 10* y consiste en:

1. El sistema parte en una posición de reposo.
2. Se realiza un hinchado del PAEC, por lo que las cavidades FREC y MREC se juntan.
3. Se hincha el MREC, esta posición da comienzo al bucle de retroceso.
4. Se hincha el DAEC y se deshincha el PAEC, manteniendo el MREC hinchado. Al encontrarse este fijado a las paredes del intestino, recoge parte del intestino que se ha movido.
5. Se hincha el FREC, de modo que se queda fijado a las paredes del intestino delgado.
6. Se deshincha el MREC.
7. Se hincha el PAEC y se deshincha el DAEC, de modo que el MREC es arrastrado hacia el FREC. Por lo que, el intestino que se había movido en el apartado 4 y fijado en el punto 5 es arrastrado hacia atrás.
8. Se hincha el MREC. En este punto se vuelve hacia el apartado 3 donde se inicia de nuevo el desplazamiento hacia atrás.

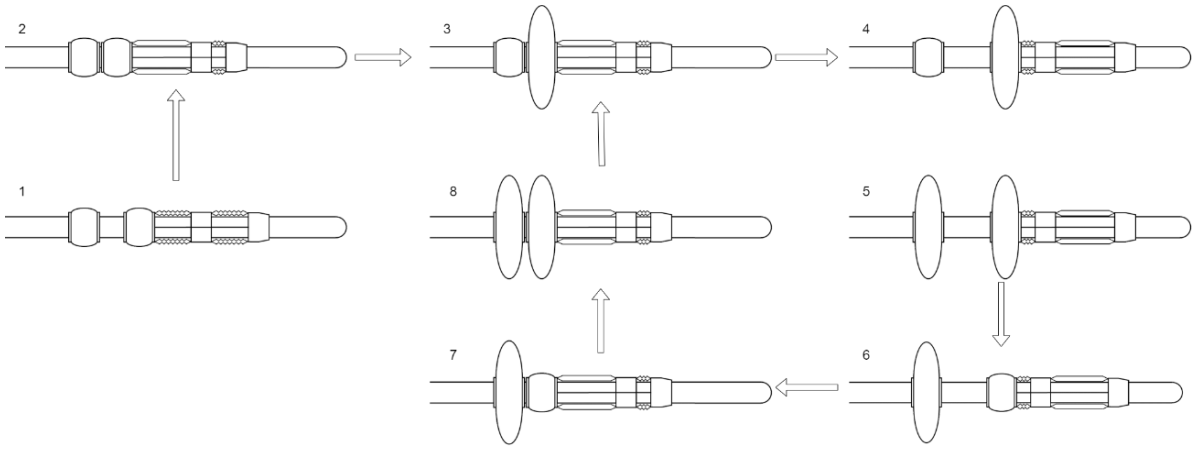


Figura 10. Secuencia de retroceso. [Elaboración propia]

Todos estos cambios han dado lugar a la versión 4.0 de Endoworm, ya que no son meras modificaciones del equipo, sino que requieren a una completa renovación del sistema.

1.3 Estructura del documento

El presente trabajo se estructura en capítulos donde de forma progresiva se adentra en cada parte relevante y que en su conjunto forman el proyecto.

En el primer capítulo se muestra una introducción de la memoria, la principal motivación, su metodología y su justificación.

El segundo capítulo detalla los objetivos principales y secundarios que existen para este trabajo final de máster.

El tercer capítulo describe la estructura que va a presentar la GUI, se detallará cada una de las pantallas existentes, como se encuentran ordenadas y como se comunica con la PCB de control.

El cuarto capítulo describe el sistema de detección de fallos empleado para las cavidades de expansión axial y radial y la metodología empleada para la obtención de las características de estas.

El quinto capítulo describe el procedimiento y los resultados obtenidos de los modelos de clasificación de señales como “Ok” o “Fuga”, mediante la aplicación de algoritmos de *Machine Learning* y que se implementarán dentro del sistema de control para la detección de fugas en tiempo real.

Con los datos del capítulo anterior se detallará en los capítulos seis y siete las conclusiones obtenidas y las futuras líneas de trabajo.

Finalmente se incluye la bibliografía, anexos donde se incluyen imágenes, y el código empleado, un presupuesto del sistema implementado y el pliego de condiciones.

Capítulo 2. Objetivos

El presente trabajo tiene como objetivos el desarrollo de una interfaz gráfica para el usuario (GUI), y el diseño de un sistema de detección de fallos de la sonda enteroscópica que sea capaz de discernir si se están hinchando o deshinchando correctamente cada una de las cavidades de expansión radial y axial del sistema Endoworm 4.0.

Para la consecución de estos objetivos se han propuesto una serie de objetivos secundarios:

- Programar la GUI según las especificaciones del médico responsable del proyecto.
- Establecer una comunicación directa entre la pantalla táctil y un ordenador, de modo que la pantalla pueda recibir y enviar información.
- Realización de ensayos donde se registran las presiones de las cavidades de la sonda enteroscópica, así como otras variables de interés para poder hacer un correcto procesado de las señales.
- Realización del segmentado de las señales de presión y la asignación de una etiqueta (ok o fuga) en función de las observaciones registradas durante los ensayos.
- Obtener las características físicas más relevantes que determinen cuando las cavidades muestran fugas de aire.
- Realizar un entrenamiento de modelos de clasificación predictivos a partir de diversos algoritmos de *Machine Learning* supervisado.
- Determinar qué tipo de modelo de *Machine Learning* es el mejor para ser incorporado en el software del dispositivo de control e implementarlo en el mismo.

Capítulo 3. Diseño e implementación de la interfaz gráfica

La interfaz gráfica, permite la comunicación entre usuario y dispositivo. Mediante esta pantalla se va a poder enviar instrucciones y recibir información relevante al estado de las cavidades.

El hardware sobre el que se va a implementar la GUI es "*Smart Embeed Display EVK101002B*", una pantalla táctil de 10.1 pulgadas en formato 16:9, que tiene como microcontrolador el STM32F746. La programación de esta pantalla se va a realizar mediante "*TouchGFX Designer - 4.18.0*", que dota de una interfaz gráfica que facilita el diseño e implementación de la GUI y autogenera el código, en C++, para cargarlo sobre el microcontrolador.

Conviene mencionar que la parte del código encargado de enviar y recibir instrucciones no se autogenera, por lo que esta parte se debe de implementar íntegramente. Sin embargo, el microcontrolador lleva implementadas todas las librerías necesarias de inicialización de cada uno de los protocolos de comunicación.

3.1 Acciones

Antes de describir como está estructurado el código y que función tiene cada pantalla conviene definir cuáles son las acciones que el usuario es capaz de seleccionar y que efecto tendrá sobre la sonda enteoroscopica, las cuales son:

- **Avanzar**: el motor neumático avanza siguiendo la secuencia de acciones definida en el apartado 1.2.2.
- **Retroceder**: el motor neumático retrocede siguiendo la secuencia de acciones definida en el apartado 1.2.2.
- **Detener**: las cavidades se mantienen en el estado en el que se encontraban antes de seleccionar esta acción.
- **Fijar**: las cavidades MREC y FREC se hinchan para que la sonda enteroscopica se quede fijada a las paredes del intestino.
- **Parada de emergencia**: todas las cavidades comienzan a deshincharse y no se podrá cambiar de acción hasta que la PCB de control indique el sistema está en condiciones para funcionar de nuevo. Esto último se hará saber al usuario mediante una desactivación del botón de parada de emergencia y la activación del botón de parada.
- **Diagnóstico**: su objetivo consiste en realizar un análisis de cada una de las cavidades y determinar si existe alguna fuga.

A parte de estas acciones existe también la posibilidad de cambiar las presiones de las cavidades.

3.2 Modelo-Vista-Presentador

La interfaz de usuario de *TouchGFX* sigue el modelo conocido como *Modelo-Vista-Presentador* (MVP) el cual es una derivación del *Modelo-Vista-Controlador* (MVC). Ambos son ampliamente empleados para aplicaciones de interfaz.

Los principales beneficios del MVP son:

- División del código, de modo que cada una de ellas tiene una función específica.
- Debido a que la parte lógica, presentador, está separado de la visualización, es mucho más fácil testear cada una de estas partes por separado.

En el modelo MVP existen tres clases:

1. **Modelo**: Esta clase siempre se encuentra activa y tiene dos propósitos:
   - Almacenar información, cada pantalla presenta su clase vista y presentador, al cambiar de pantalla la información que puede haber en estas dos clases es eliminada. Este es uno de los objetivos de esta clase, guardar información relevante entre cambios de pantalla.
   - Actuar como una interfaz hacia el sistema *backend*, transmitiendo eventos hacia y desde la pantalla actualmente activa.

   La clase modelo se configura automáticamente para tener un puntero al presentador que actualmente se encuentra activo. Cuando se producen cambios en el modelo, se notifica el cambio a la pantalla que está activa. Esto se consigue a través de métodos en la interfaz *ModelListener* en la aplicación.

2. **Vista**: Es una interfaz pasiva que se encarga de mostrar los datos que puede haber en el modelo. También se definen los elementos que interactúan con el usuario, como pueden ser los botones, barras, imágenes, etc… En esta clase no se llegan a indicar que se tiene que hacer cuando el usuario, por ejemplo, aprieta un botón, esta lógica la implementa el presentador.

3. **Presentador**: Es el responsable de la lógica que va a seguir la vista. Recibe eventos tanto del *backend* del modelo como de la interfaz de usuario de la vista y decidirá qué acción tomar. Por ejemplo, si se recibe un evento de alarma por parte del modelo, el presentador, puede decidir que se tiene que ver una ventana emergente de alarma en la vista de la pantalla.

   Llamaremos *backend* a la parte de la aplicación que no forma parte de la interfaz gráfica y la comunicación con este se realiza desde la clase modelo. Desde el sistema *backend* se pueden llegar a enviar eventos que pueden desencadenar acciones y cambiar la vista de la interfaz gráfica y también desde la vista se pueden enviar eventos al *backend* para llevar a cabo alguna acción. En el presente trabajo, el *backend* sería el microcontrolador que controla el Endorworm, por ejemplo, desde la pantalla se pueden enviar la instrucción *"Stop"* que movería la máquina de estados al estado *"Parar"* o el microcontrolador podría informar que alguna cavidad esta dañada a la pantalla y de este modo alertar al usuario. El modelo MVP se puede apreciar en la *Figura 11*.

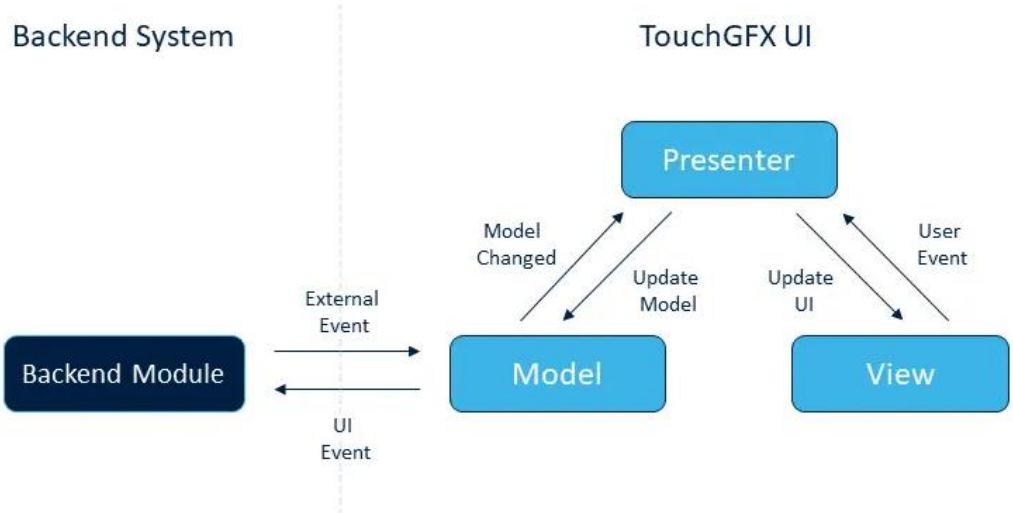


Figura 11 Modelo-Vista-Presentador y comunicación externa [19].

3.3 Inicio

La pantalla de inicio es la que se observa al encender la pantalla, tiene como objetivo esperar a que el resto de las configuraciones del equipo de Endoworm estes listas para comenzar a funcionar. Mientras el sistema se está configurando, esta pantalla va mostrando una barra circular que se va llenando cada cierto tiempo para indicar al usuario que debe de esperar. Una vez finalizado, esta pantalla no se volverá a mostrar durante el flujo normal de funcionamiento del equipo.

Figura 12. Pantalla de inicio. [Elaboración propia]

3.4 Menú

La pantalla menú se puede observar en la *Figura 13*. Tal como se observa, aparece una imagen en el centro de la pantalla, esta imagen representa el estado actual de las cavidades y se irá actualizando conforme el sistema empiece a moverse, y también se puede observar un total de 5 botones.

El botón menú es un desplegable mediante el cual se puede acceder al resto de pantallas, en el resto de las pantallas también está incluido este botón, pero solo se permite acceder a la pantalla menú. El resto de los botones, tal como indican sus respectivos nombres, implementan las acciones de hacia delante, parada, fijación, hacia atrás y parada de emergencia sobre las cavidades.

Además, cabe destacar que cuando el equipo entra por primera vez a esta pantalla, la acción que va a haber por defecto es la de parada ya que esta consiste la acción de reposo del sistema.

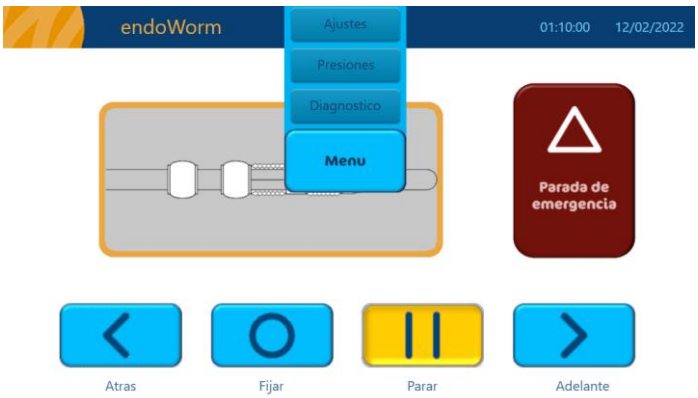


Figura 13. Pantalla Menú. [Elaboración propia]

3.5 Ajustes

La siguiente pantalla por comentar es la de ajustes, la cual se puede observar en la *Figura 14*. En esta pantalla, la única acción que está implementada es la de brillo, mediante la cual se puede ajustar el brillo de todas las pantallas. El resto de las acciones se desarrollarán más adelante, después del presente Trabajo Final de Máster.

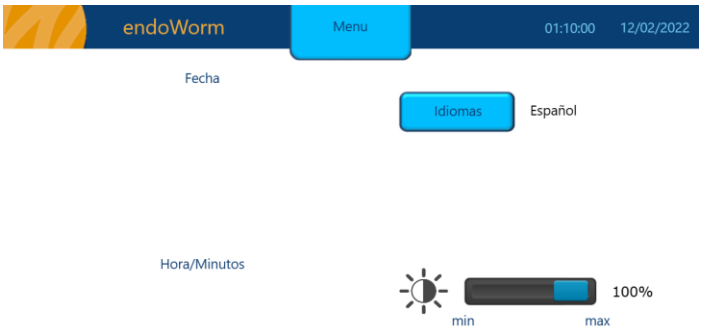


Figura 14. Pantalla Ajustes. [Elaboración propia]

3.6 Presiones

La pantalla de presiones se puede observar en la *Figura 15*. El objetivo de esta pantalla consiste en modificar los valores de las presiones que van a haber en cada una de las cavidades, los valores presentan un rango de 0 hasta 9. Tal como se observa existen tres barras, en vez de cuatro para cada una de las cavidades, esto es debido a que se ha decido ajustar el DAEC y PAEC con la misma barra, ya que se llegó a la conclusión de que estas dos cavidades no tendrían sentido modificarlas por separado. Además, el nombre que tienen asignado es distinto, esto es debido a que los nombres MREC, FREC, PAEC y DAEC, podrían ser confusos de cara al usuario y difíciles de recordar, por lo que se decido emplear nombres más descriptivos. Empezado por la izquierda, se tiene "*fixed ballon*", el cual representa el FREC, que indica la cavidad que está fijada, después "*mobile ballon*", que hace referencia al MREC, el cual es el globo de expansión radial móvil y finalmente los "*bellows*", el cual representa el DAEC y PAEC.

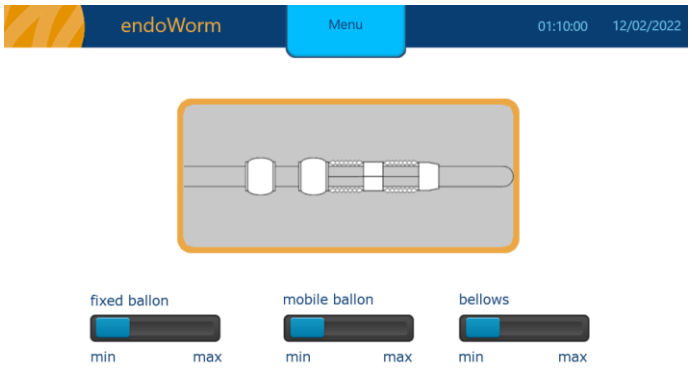


Figura 15. Pantalla de Presiones. [Elaboración propia]

3.7 Diagnostico

La última pantalla que queda por comentar es la de Diagnóstico, cuando el usuario ha seleccionado esta opción, la pantalla se queda bloqueada y el usuario podrá observar que se está ejecutando esta acción mediante el movimiento de una barra que se va llenando y vaciando. Cuando termine, se mostrará una imagen de las cavidades, resaltando en rojo en caso de alguna tenga una fuga y un texto debajo detallando cual es problema. En la *Figura 16* se puede observar un ejemplo de la acción diagnóstico indicando cual es el resultando.

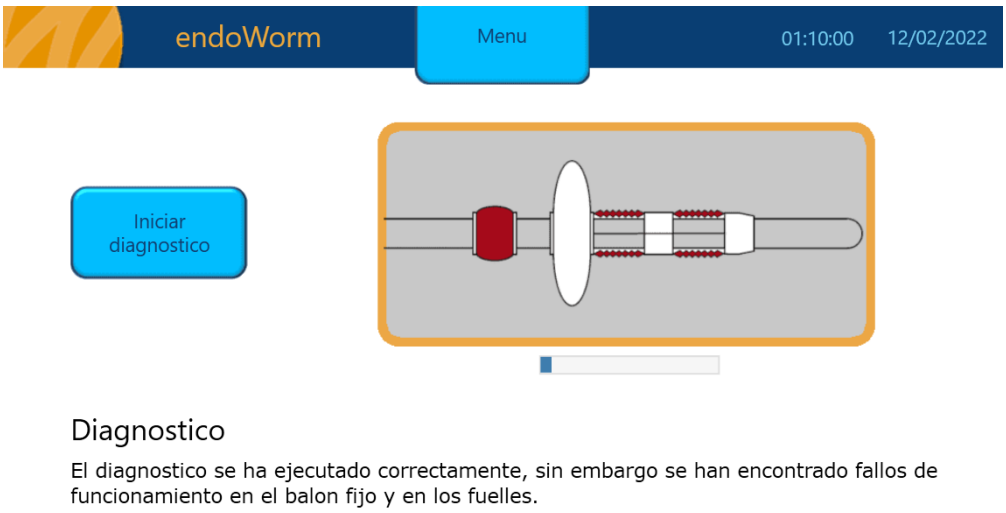


Figura 16. Pantalla de Diagnóstico mostrando un fallo en FREC PAEC y DAEC. [Elaboración propia]

3.8 Modelo

Una vez explicado cada una de las pantallas se va a dar una explicación sobre el modelo. Tal como se ha indicado en el apartado 3.2Modelo-Vista-Presentado, el modelo es aquella parte del código donde se almacenan los datos que se tienen que mantener entre los cambios de pantalla y además donde se envía los eventos generados por parte del usuario que tienen como destino la placa de control y viceversa.
La placa de control y la pantalla táctil se pueden comunicar de varias formas, entre ellas se encuentran: RS485, CAN, RS232, I2C, SPI y USB [20]. Entre estos se ha elegido el RS232 al ser un sistema de comunicación eficiente y libre de ruido, se podría haber empleado el RS485, como otra opción, sin embargo, el RS232 es full-dúplex, es decir puede transmitir en ambas direcciones, mientras que el RS485 es half-dúplex, solo puede transmitir en una dirección. Se ha dejado abierta la posibilidad de cambiarla a un futuro a comunicaciones USB.

La comunicación entre pantalla y placa de control se diferencia en dos partes. En primer lugar, a la hora de iniciar el equipo, la primera pantalla que se va a observar es la de Inicio, tal como se mencionó en el apartado 3.3 Inicio, y se sale de esta cuando el equipo está listo para usarse, esto se traduce en enviar un ‘1’ a la pantalla. Esta parte del código se puede observar en la *Figura 17*.

```
if(_RS232RevSt.RevF==true)
{
    _RS232RevSt.RevF=false;

    /*  The first time should execute the if condition to change to menu screen */
    if(true == FirstFlag)
    {
        /* With this line anything, besides 0 that the RS232 is going to send will be translated to 1,
         * which indicates change of screen */
        if(1 == _RS232RevSt.pdata[0])
        {
            /* The first data indicates if the device is ready to work */
            modelListener->ChangeScreenInicioView(_RS232RevSt.pdata[0]);
            FirstFlag = false;
        }
    }
```

Figura 17. Configuración de pantalla de Inicio. [Elaboración propia]

Una vez que el equipo está listo para usarse, la placa de control espera a que el usuario pulse alguna de las opciones. Cuando se seleccione la acción deseada el equipo empezará a funcionar y la placa de control enviará hacia la pantalla dos datos, en formato hexadecimal y en el siguiente orden: la acción que se está realizando y el estado de las cavidades.

La acción se representa desde el número 0 hasta el 5. Estos números simbolizan, en este orden: *Stop*, *Forward*, *Backward*, *Fix*, *Emergency* y *Diagnosis*. Cabe destacar que desde la pantalla se puede enviar cualquier acción a la placa de control, pero la placa de control solo puede enviar a la pantalla la acción de *Stop*. Esta orden es enviada a la pantalla cuando el usuario ha apretado anteriormente la acción de *Emergency*. Cuando se aprieta este botón, este se queda clavado y se bloquean el resto de botones, el desbloqueo es automático, pero no se producirá hasta que se vacíen por completo todas las cavidades de la sonda enteroscopica. La forma que tiene la placa de control de comunicarlo al usuario es soltando el botón de *Emergency* y seleccionado el botón de *Stop*.

Por otro lado, el número que codifica cada una de las imágenes correspondientes al estado de las cavidades es de tipo entero sin signo de 8 bits. Los primeros 4 bits menos significativos representan el hinchado, 1, y el deshinchado, 0, de cada una de las cuatro cavidades que conforman la sonda. Siendo el bit 3 el FREC, el bit 2 el MREC, el bit 1 el PAEC y el bit 0 el DAEC. Mientras que los 4 bits más significativos representan si alguna cavidad está dañada, 1 significa que está dañado y 0 que funciona correctamente, estos cuatro bits presentan el mismo orden que los menos significativos. A modo de ejemplo se puede mostrar la *Figura 18*.

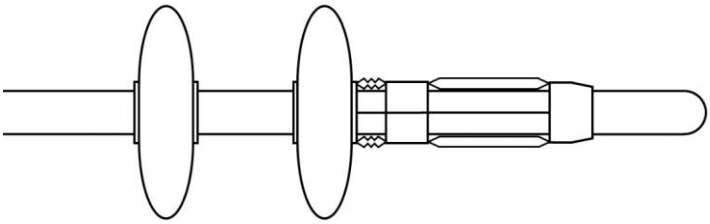


Figura 18. Ejemplo de formato de las cavidades. [Elaboración propia]

En la *Figura 18* se puede observar que las cavidades que están hinchadas son el FREC, el MREC y el DAEC y ninguna de ellas está dañada. Por lo tanto, tendría el siguiente formato: 0000 1101, el cual en hexadecimal es 0x0D. Siguiendo con este ejemplo, si el usuario hubiera seleccionado la opción de Avanzar, el paquete de datos que se enviaría a la pantalla sería 0x01 seguido de 0x0D.

La parte del código encargado de gestionar estos dos datos se observa en la *Figura 19*.

```
//Equivale al estado del cavitie state
switch (_RS232RevSt.pdata[0])
{
    case 0:
        button = Stop;
        break;

    case 1:
        button = Forward;
        break;

    case 2:
        button = Backward;
        break;

    case 3:
        button = Fix;
        break;

    case 4:
        button = Emergency;
        break;

    case 5:
        button = Diagnosis;
        break;

    default:
        button = Stop;
        break;
}

modelListener->RS232ChangeRatioButtonState(button);

/* Updates the state of diagnostic screen */
if(true == isDiagnostic)
{
    if(Stop == _RS232RevSt.pdata[0])
    {
        animation = dictionary_fire(table_state_animations, _RS232RevSt.pdata[1]);
        modelListener->RS232DiagnosticResult(animation);
        // Set to false to avoid changing diagnostic status without calling to the function
        // that starts the Diagnostic
        isDiagnostic = false;
    }
}
else
{
    // Translator's execution
    // Se recepciona el valor de "cavities_state" del uC de la PCb de control
    animation = dictionary_fire(table_state_animations, _RS232RevSt.pdata[1]);
    modelListener->RS232ChangeImageMenu(animation);
}
```

Figura 19. Descomposición del dato de entrada. [Elaboración propia]

Esta parte del código se ejecuta cada 10ms y tal como se observa en la *Figura 19* existen dos partes. En primer lugar, mediante el *switch-case* se determina la acción que se ha seleccionado, seguido de una condición *if-else*, el cual, dependiendo de la acción seleccionada, actualizará la imagen de la pantalla Menú o la de Diagnóstico. En el Anexo A1. Código de pantalla se puede ver el código completo. Cabe destacar que, en el caso de la acción de Diagnóstico, el resultado de este proceso se enviará mediante un paquete de datos donde el primer dato será un 0, Detener, seguido del estado de las cavidades.

Por otro lado, cada vez que se seleccione una acción se enviará a la placa de control un paquete de datos formado por la acción a realizar y los valores de presiones del FREC, MREC y los del PAEC y DAEC, respectivamente, tal como se había mencionado en el apartado 3.6 Presiones. A modo de ejemplo se puede observar la *Figura 20*, donde el usuario ha seleccionado la acción de Avanzar y los valores de presiones de cada una de las cavidades.

1	5	4	5
acción	FREC	MREC	PAEC y DAEC

Figura 20. Ejemplo de paquete de datos a enviar a la placa de control, donde 1 es la acción que se desea y 5,4 y 5 las presiones de FREC, MREC y PAEC y DAEC, respectivamente.

En la *Figura 21* se puede observar el código encargado de enviar los datos.

```
void Model::usart_tx(uint8_t *data)
{

    #ifndef SIMULATOR

     /* This determines if the last operation was a diagnostic */
     isDiagnostic = (data[0] == 5)?true:false;

     n = (uint16_t) sizeof(data)/sizeof(uint8_t);
     EDT_UART_Transmit_IT(&hRs232, data, n);
    #endif
}
```

Figura 21. Envió de datos a la placa de control. [Elaboración propia]

Capítulo 4. Sistema de Machine Learning de detección de fallos

El segundo objetivo que se plantea para este TFM consiste en realizar un sistema de detección de fallos para los dos tipos de cavidades que se encuentran en la sonda enteroscopica, AEC y REC.

La razón por la que se va a desarrollar este sistema es debido a la dificultad existente para el especialista de determinar si el equipo está fallando y, en caso de hacerlo, cuál de todas las cavidades presenta el fallo. Es por esto, que el desarrollo de un sistema autónomo para la detección de fugas de aire a partir del hinchado y deshinchado de las cavidades es necesario. La detección de fugas es una parte primordial ya que un mal funcionamiento del equipo podría poner en riesgo al paciente e incluso se traduciría en una perdida significante de la eficiencia del dispositivo en el avance a lo largo del intestino delgado [21].

Se han realizado varios estudios para la detección automática de fugas en distintas áreas industriales, por ejemplo, para detectar fugas de aire en tuberías [22], y en algunos de estos casos se han usado métodos de Machine Learning [23]. Sin embargo, a diferencia de estos estudios, las cavidades que presenta el sistema Endoworm 4.0 se deforman, por lo que no se puede evaluar la presencia o ausencia de una fuga simplemente observando si su presión ha variado en algún instante. En su lugar, se debe de observar la evolución de las señales de presión, de las cavidades, y evaluar a partir de estas si es posible que haya fugas de aire.

En aplicaciones médicas, los modelos de clasificación predictivos son comúnmente empleados para solucionar casos binarios, es decir de dos opciones, con ratios de rendimiento altos [24]. Estas técnicas pueden ser usadas para clasificar el estado de cada cavidad como "*correcto*" o "*fuga*", mientras el equipo está funcionando.

Para implementar este sistema dentro del dispositivo de control, en primer lugar, se debe de realizar un análisis a partir de los datos obtenidos gracias a la lectura de los sensores de presión que se encuentran conectados a cada una de las cavidades. A partir de estas se estudiarán cada una de las características que se puedan obtener de estas señales y se seleccionarán aquellas que puedan determinar con mayor exactitud si una cavidad está funcionando correctamente de una que no lo está.

Tanto el procesado digital de las señales como el entrenamiento de los modelos de *Machine Learning* se han realizado mediante la herramienta *Matlab R2022b*, bajo la licencia adquirida por la *Universidad Politécnica de Valencia*.

4.1 Obtención de las señales y características

Para los modelos de clasificación predictivos que se tienen que obtener, se requiere una serie de entradas (características) que contienen la información necesaria de las señales de presión de las cavidades, para clasificar cada uno de los casos como "*correcto*" o "*fuga*".

El punto de partida para obtener cada una de las características es la presión absoluta recibida por parte de los sensores. Existen un total de seis señales: la presión de llenado y vaciado del sistema, la presión de las dos cavidades de expansión axial y las otras dos de expansión radial. De estas seis señales las únicas que son importantes son las que hacen referencia a las cavidades. Por lo tanto, estos servirán como fuentes de información de las cuales se extraerá los datos necesarios en forma de características o parámetros.

Para obtener estas señales se han llevado a cabo varios ensayos, en los que se ha ido modificado los tiempos de presión de las cavidades, para obtener la mayor variabilidad posible. En alguno de estos ensayos el sistema Endoworm se podía mover libremente (al aire libre) y en otros se

encontraba en un intestino artificial hecho de PU (poliuretano) con un diámetro variable de entre 25mm a 35mm, mientras estaba siendo manipulado por un especialista.

La decisión de si una cavidad es "*correcto*" o "*fuga*" fue determinada por la persona que estaba realizando el experimento, observando si en algún instante de los ciclos de hinchado y deshinchado de cada cavidad aparece alguna fuga de aire.

A partir de cada registro se extrajo una conjunto de señales, como las que se pueden observar en la *Figura 22*.

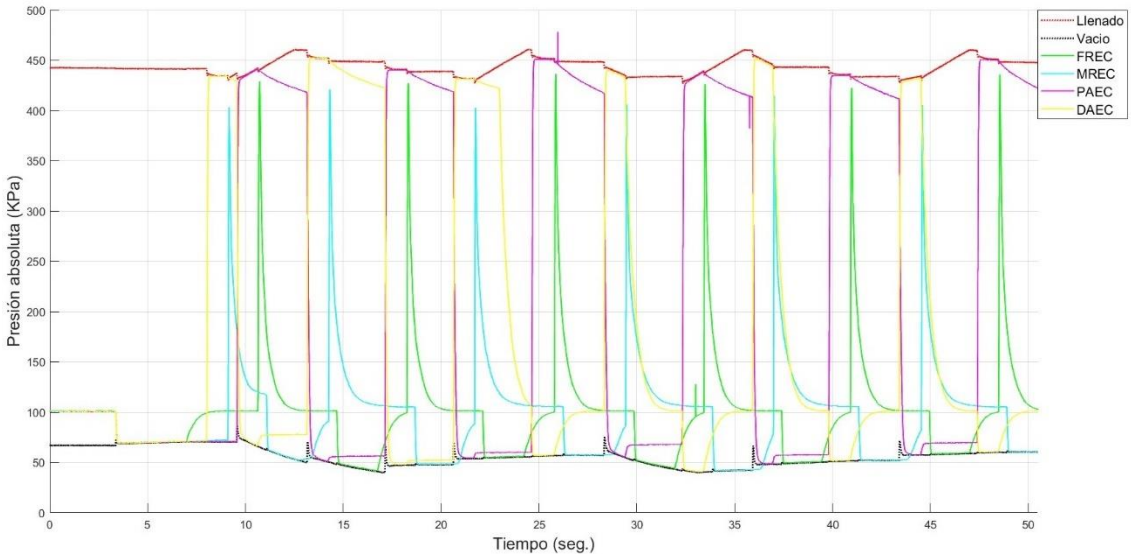


Figura 22. Representación temporal de las cavidades [Elaboración propia]

Junto a estas señales también se obtuvieron los pulsos generados por las electroválvulas de llenado y vaciado de cada una de las cavidades y las acciones que se estaban desarrollando en cada instante.

Una vez que se han obtenido las señales de todas las cavidades, el siguiente paso consiste en segmentarlas. La segmentación está formada por dos paso: en primer lugar, determinar la acción que nos interesa y en segundo lugar establecer la ventana de cada una de las señales segmentadas. La acción en la que se va a centrar el presente trabajo es el de avanzar, pero en el futuro se deberán de analizar el resto de las acciones. Por otro lado, se ha establecido que la ventana tiene como punto inicial 0,5 milisegundos antes del flanco de subida de la electroválvula de llenado y como punto final 0,5 milisegundos después del flanco de bajada de la electroválvula de vaciado. Esta ventana se eligió de modo que se viera la totalidad de cada una de las señales segmentadas y ver además cómo se encontraban las señales de presión antes y después de la acción de las electroválvulas.

En las *Figuras 23* y *24* se muestran dos ejemplos de segmentación uno por cada tipo de cavidad, mostrando los pulsos de subida y bajada de las electroválvulas.

Además, apoyándose en esta figuras, se van a definir dos regímenes: el estacionario y transitorio, los cuales se emplearán a la hora de detallar las características de las señales. En el caso de las señales tipo AEC, *Figura 23*, el régimen estacionario se define desde la detección del flanco de bajada de la electroválvula de llenado hasta la muestra anterior a la detección del flanco de subida de la electroválvula de vaciado. Por otro lado, en el REC, *Figura 24*, este régimen comienza 1,5 segundos después de detectar el flanco de bajada de la electroválvula de llenado, hasta la muestra anterior a la detección del flanco de subida de la electroválvula de vaciado. En ambos casos, el

régimen transitorio comienza con la detección del flanco de subida de la electroválvula de llenado hasta su flanco de bajada.

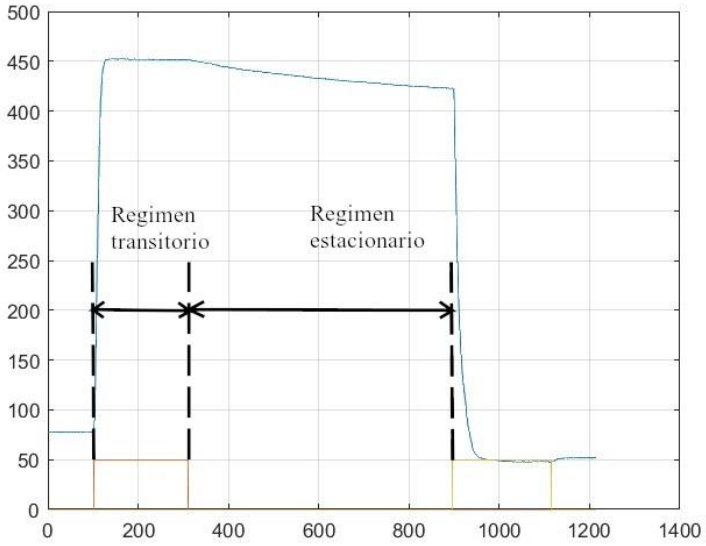


Figura 23. Representación de las señales AEC y los pulsos de las electroválvulas (líneas naranja y rojo) [Elaboración propia]

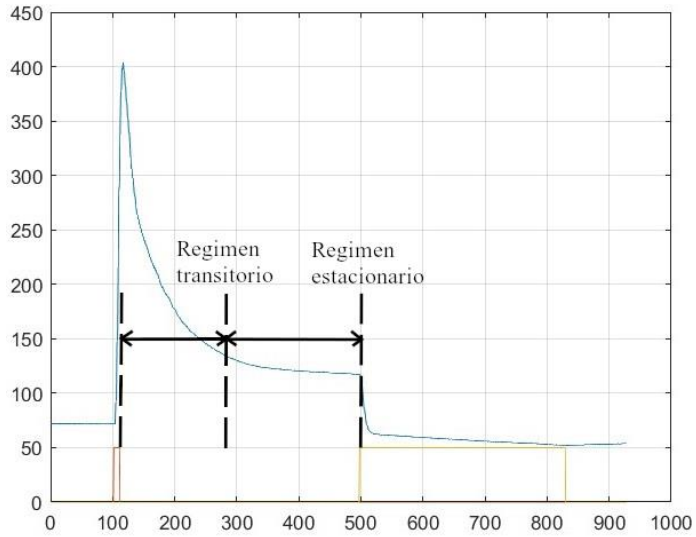


Figura 24. Representación de las señales REC y los pulsos de las electroválvulas (líneas naranja y rojo) [Elaboración propia]

Una vez que se han obtenido todas las señales segmentadas, el siguiente paso consiste en obtener las características que sean capaces de reflejar aquellos aspectos que permitan diferenciar entre una señal con y sin fuga. Además, con el objetivo de etiquetar las señales se ha decido llamar a las características de los globos de expansión radial como "*a*" y para los de expansión axial "*b*". Las características que se van a extraer para los dos tipos de señal son:

- La presión atmosférica (a_1 y b_1).

- Presión media (a_2 y b_2), desviación típica de la presión (a_3 y b_3), diferencia de presión entre el punto inicial y final del estado estacionario (a_4 y b_4) y la pendiente en el estado estacionario (a_5 y b_5).
- Desde el principio de la señal segmentada hasta el flanco de subida de la electroválvula de llenado, la desviación típica de la presión (a_6 y b_6), la media de presión (a_7 y b_7) y la diferencia de presión entre el punto inicial y final (a_8 y b_8).
- Desde el flanco de bajada de la electroválvula de vaciado hasta el final de la señal segmentada la desviación típica de la presión (a_9 y b_9), la media de presión (a_{10} y b_{10}) y la diferencia de presión entre el punto inicial y final (a_{11} y b_{11}).
- Tiempo total de la señal segmentada (a_{12} y b_{12}), del pulso de la electroválvula de llenado (a_{13} y b_{13}) y vaciado (a_{14} y b_{14}).

A parte estas, en el caso del AEC se obtendrá la media del pulso generado por la electroválvula de llenado (a_{15}). Y en el caso de las cavidades REC se van a obtener tres pendientes que determinan el estado transitorio de la señal (b_{15}, b_{16} y b_{17}).

Por otro lado, con el objetivo de optimizar el rendimiento de los algoritmos y hacer el entrenamiento y validación mucho más eficiente, se va a realizar la normalización de las características empleando el método de *z-score*. Este método describe el número de desviaciones estándar por el cual un dato está por encima o por debajo de la media de los datos que se han observado [25]. La fórmula que describe este método es:

$$Z = \frac{X - \mu}{\sigma} \qquad (1)$$

La fórmula (1) viene a indicar que una vez que se tengan todos los datos, se calculará la media (μ) y desviación típica (σ) y a partir de estos se calculará para cada dato (X) su normalizada (Z). Estos datos normalizados son los que se usarán para el entrenamiento de los modelos de *Machine Learning* y también se exportarán al microcontrolador para poder normalizar los datos que vengan del exterior.

4.1.1 Selección de características

El número de combinaciones y características que se pueden llegar a realizar es muy elevado por lo que para reducir el número de características se van a aplicar cuatro filtros para que únicamente queden aquellas que tiene mayor influencia a la hora de determinar si una señal presenta una fuga o no, estos filtros son:

- **Fisher's score**: Se calcula mediante la siguiente ecuación (2):

$$F(k) = \frac{\sum_{i=1}^{c} n_i \left(\mu_k^i - \mu_k\right)^2}{\sum_{i=1}^{c} n_i \left(\sigma_k^i\right)^2} \qquad (2)$$

  En el que n_i es el número de las i-clases, μ_k^i y σ_k^i son la media y la varianza de la característica k en la clase i, respectivamente, y μ_k es la media de la característica k en todas las clases [26].

- **ReflieF**: Este tipo de filtro supone que una característica es muy relevante si es capaz de diferenciar entre dos instancias de diferentes clases, y mediante esta lógica se define un peso para cada característica [27].

- **Chi-square**: Este tipo de filtro confronta las conclusiones observadas en una investigación con una serie de resultados teóricos. Estos últimos resultados obtenidos se calcularon asumiendo la independencia de estas variables. La diferencia entre los resultados observados y los supuestos se condensa en el valor que recibe el estadístico x^2,

el cual lleva asociado, un valor-p. A partir de este, se acepta o deniega la hipótesis de independencia de variables: si es menor que un umbral fijado se rechaza, de lo contrario se acepta[28].

- **MRMR score**: El algoritmo MRMR se encarga de buscar un subconjunto de características de los datos que sea capaz de clasificar lo mejor posible todas las clases mientras que, las características que lo componen son lo más diferentes posibles entre sí [29].

A la hora emplear cada uno de estos filtros para todas las características se obtiene una puntuación y una posición en un ranking, cuanto más bajo esté una característica en el ranking mejor, ya que significa que ha obtenido un buen resultado.

Una vez que se aplican todos los filtros, cada característica obtendrá cuatro puntuaciones y cuatro posiciones en cuatro rankings distintos, uno por cada filtro aplicado. Estas cuatro posiciones se suman para obtener una posición en un ranking final donde se engloban los cuatro filtros. A partir de este ranking se eligen las diez mejores características para cada tipo de cavidad, descartando el resto [3].

4.1.2 Señales y características obtenidas

La totalidad de señales segmentadas que se han obtenido es de 1336 para el tipo balón y 1369 de fuelles, a cada una de ellas se les ha aplicado cuatro filtros, los cuales se han definido en el apartado 4.1.1, y a partir de los resultados obtenidos se han seleccionado las diez que mejor puntuación presentan.

En la *Tabla 3* y *4* se muestran los resultados de los dos tipos de cavidades. Tal como se observa en la *Tabla 3* las características que mejor puntuación han obtenido son: $a_1 - a_8$, a_{11} y a_{12}. Respecto a la *Tabla 4*, las características $b_1 - b_9$ y b_{11} son las que han presentado un mejor resultado.

Características		Puntuación				Puntuaciones normalizadas				Ranking				Ranking final	
		Fisher's	ReliefF	Chi2	MRMR	Fisher's	ReliefF	Chi2	MRMR	Fisher's	ReliefF	Chi2	MRMR	Puntos	Final
Cavidad radial	a1	0,022	0,067	89,448	7,53E-02	-0,960	0,567	-0,630	1,99E-02	2	2	2	2	8	1
	a2	1,706	0,106	254,430	2,48E-01	2,986	1,849	1,536	2,29E+00	7	7	7	3	24	4
	a3	0,377	0,056	220,096	2,11E-01	-0,129	0,209	1,085	1,81E+00	8	5	3	10	26	5
	a4	0,420	0,059	191,160	3,23E-02	-0,026	0,301	0,705	-5,48E-01	5	1	8	6	20	3
	a5	0,431	0,103	187,123	4,58E-02	-0,003	1,775	0,652	-3,69E-01	4	15	4	7	30	7 ó 8
	a6	0,398	0,021	188,668	1,29E-01	-0,080	-0,936	0,673	7,25E-01	6	4	6	1	17	2
	a7	1,281	0,104	232,434	1,21E-01	1,989	1,780	1,247	6,23E-01	11	3	5	9	28	6
	a8	0,439	0,025	205,650	2,81E-02	0,017	-0,786	0,896	-6,03E-01	3	17	11	5	36	9 ó 10
	a9	0,366	0,040	155,462	5,93E-02	-0,155	-0,304	0,237	-1,91E-01	9	9	9	17	44	11 ó 12
	a10	0,325	0,013	88,814	1,77E-01	-0,249	-1,206	-0,638	1,36E+00	17	14	17	4	52	14
	a11	0,395	0,031	169,883	1,85E-02	-0,086	-0,605	0,426	-7,30E-01	10	11	1	8	30	7 ó 8
	a12	0,043	0,030	17,227	1,22E-02	-0,910	-0,645	-1,578	-8,14E-01	16	12	10	15	53	15 ó 16
	a13	0,154	0,020	35,798	4,02E-03	-0,650	-0,977	-1,334	-9,21E-01	15	16	16	16	63	17
	a14	0,157	0,031	34,964	1,15E-02	-0,644	-0,593	-1,345	-8,23E-01	14	8	15	11	48	13
	a15	0,237	0,061	69,448	2,26E-02	-0,456	0,375	-0,893	-6,76E-01	13	6	13	12	44	11 ó 12
	a16	0,262	0,027	69,947	2,05E-02	-0,397	-0,730	-0,886	-7,04E-01	12	13	14	14	53	15 ó 16
	a17	0,325	0,047	125,866	3,98E-02	-0,249	-0,074	-0,152	-4,49E-01	1	10	12	13	36	9 ó 10

Tabla 3. Resultados de aplicar los filtros a las características del REC [Elaboración propia]

Características		Puntuación				Puntuaciones normalizadas				Ranking				Ranking final	
		Fisher's	ReliefF	Chi2	MRMR	Fisher's	ReliefF	Chi2	MRMR	Fisher's	ReliefF	Chi2	MRMR	Puntos	Final
Cavidad axial	b1	0,617	0,084	143,626	2,64E-14	-0,316	-0,020	-0,279	-3,77E-01	4	5	5	3	17	2
	b2	2,200	0,098	399,912	2,06E-14	0,790	0,344	1,396	-3,77E-01	3	15	4	15	37	10
	b3	3,785	0,085	406,240	3,77E-01	1,897	0,002	1,437	2,70E+00	2	7	3	6	18	3 ó 4
	b4	4,478	0,103	409,150	1,44E-14	2,382	0,489	1,456	-3,77E-01	5	4	2	7	18	3 ó 4
	b5	1,797	0,163	418,252	2,46E-14	0,509	2,091	1,515	-3,77E-01	7	14	7	1	29	7
	b6	0,174	0,081	123,561	6,23E-14	-0,625	-0,098	-0,410	-3,77E-01	1	2	8	5	16	1
	b7	1,603	0,122	270,274	4,06E-14	0,373	0,979	0,549	-3,77E-01	15	3	10	2	30	8
	b8	0,267	0,068	158,713	1,23E-14	-0,560	-0,463	-0,180	-3,77E-01	10	10	1	4	25	5 ó 6
	b9	0,143	0,048	65,952	7,62E-15	-0,647	-0,984	-0,787	-3,77E-01	8	1	6	10	25	5 ó 6
	b10	0,337	0,085	145,053	1,32E-14	-0,511	-0,007	-0,270	-3,77E-01	11	6	15	8	40	11
	b11	0,217	0,049	82,830	3,13E-15	-0,595	-0,972	-0,676	-3,77E-01	6	8	11	9	34	9
	b12	0,002	0,049	29,086	1,29E-27	-0,745	-0,966	-1,028	-3,77E-01	9	12	9	11	41	12
	b13	0,021	0,012	11,065	3,96E-16	-0,732	-1,937	-1,145	-3,77E-01	13	11	12	13	49	13
	b14	0,013	0,099	13,709	3,68E-16	-0,737	0,382	-1,128	-3,77E-01	14	9	14	14	51	15
	b15	0,378	0,129	117,534	3,17E-01	-0,483	1,160	-0,450	2,21E+00	12	13	13	12	50	14

Tabla 4. Resultados de aplicar los filtros a las características del AEC [Elaboración propia]

A pesar de que con los filtros se obtiene unos resultados bastante concluyentes, es buena idea obtener la representación de estas características en forma de cajas y bigotes, tal como se observan en las *Figuras 25* y *26*. Este tipo de gráficas son muy útiles ya que mediante un simple vistazo se puede determinar si una característica es capaz de diferenciar cuando una cavidad presenta una fuga o no, ya que, en caso de ser capaz de hacerlo las cajas y bigotes entre los dos casos no tendrían ninguna parte superpuesta, es decir, no habría ningún dato que estuviera al mismo nivel en ambos casos. Tal como se observan en las *Figuras 25* y *26*, las características seleccionadas a pesar de ser las que mejor puntuación han obtenido no son del todo capaces de diferenciar por sí solo, los casos de señal correcta o con fuga, por lo tanto, será necesario que se empleen una combinación de estas características para el entrenamiento de los modelos de *Machine Learning.*

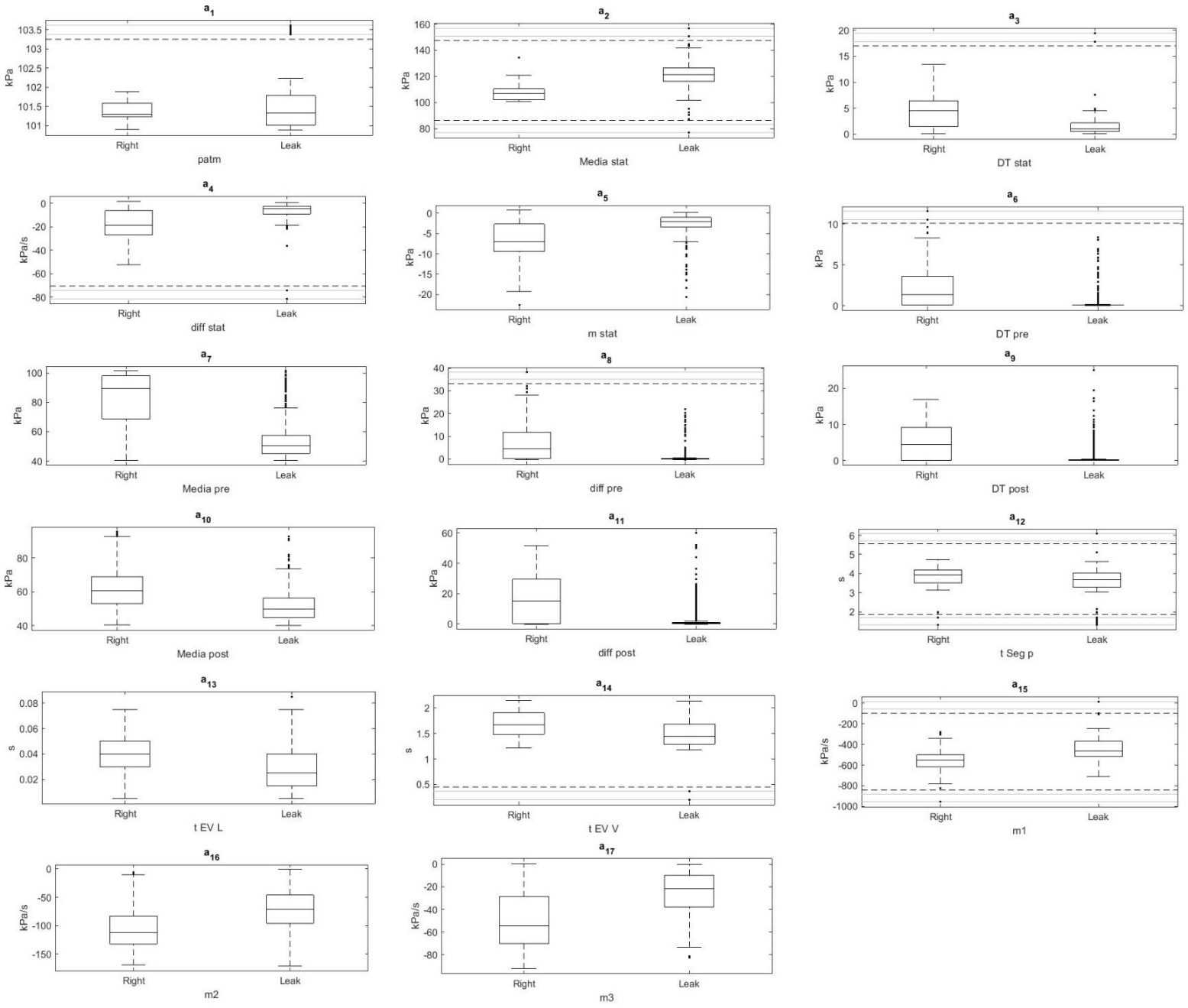


Figura 25. Características REC

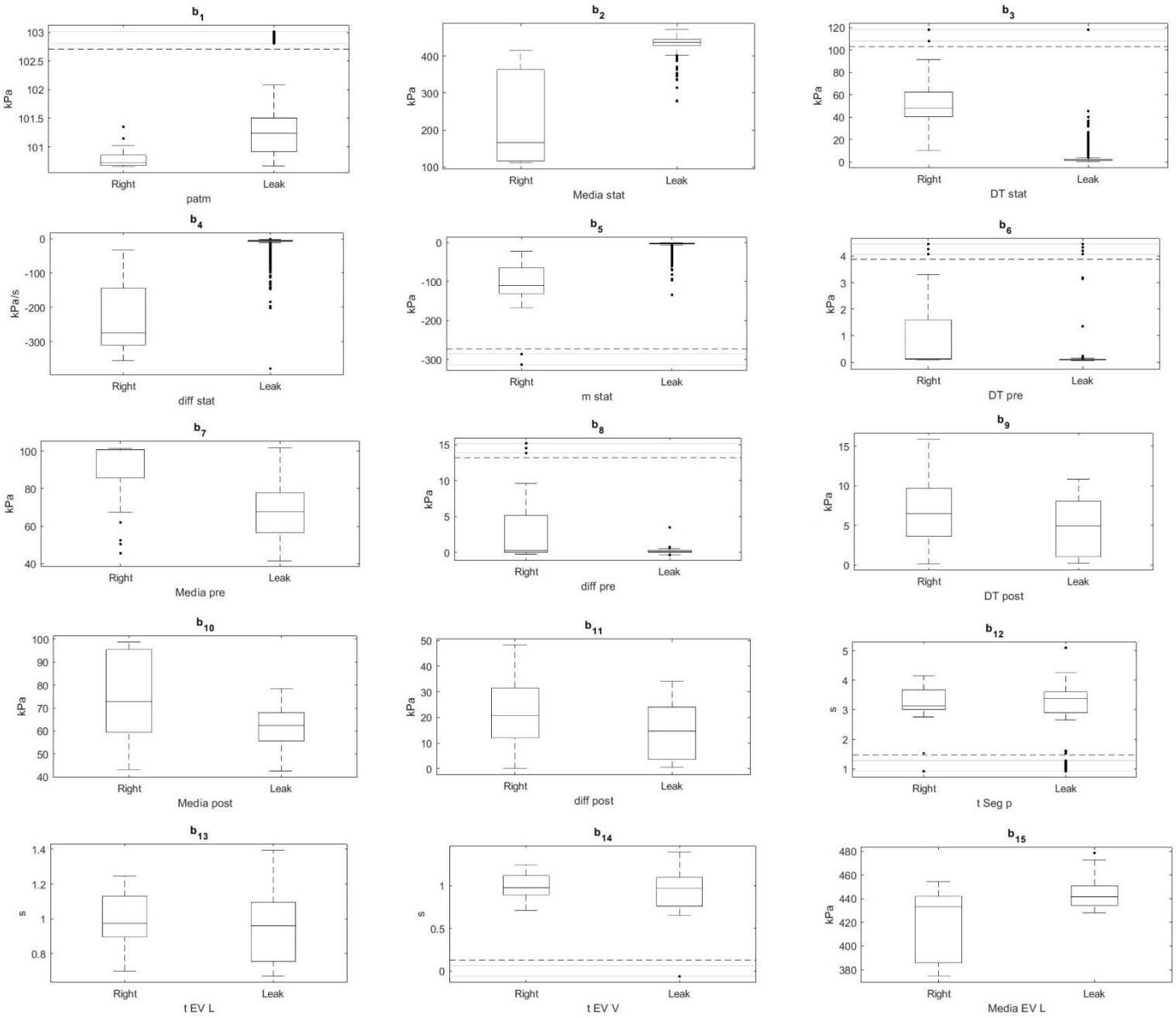


Figura 26. Características AEC

4.2 Entrenamiento y validación de modelos de Machine Learning

Una vez determinado cuales son las características que se van a emplear, el siguiente paso consiste en entrenar los modelos de *Machine Learning*, para ello se ha utilizado la herramienta *Classification Learner* de *Matlab R2022b*. Otra alternativa a esta herramienta es la APP *Deep Learning* la cual nos permite desarrollar redes neuronales e implementarlos directamente en una placa de STM32 desde la interfaz CubeMX, integrada en el entorno de desarrollo de STM32. Esta APP se decidió no usarla ya que es exclusiva de redes neuronales, las cuales pueden llegar a ocupar bastante memoria y tiempo de ejecución, parámetros que se intentan optimizar para mejorar la ejecución en tiempo real sobre el microcontrolador.

Sin embargo, antes de comenzar a entrenar los modelos hay que separar los datos en dos grupos: el primer grupo se encarga de entrenar y validar los modelos, para ello se ha empleado el 80% de los datos que se han extraído, y el segundo grupo se encarga de testearlos, el 20% de restante, para determinar cómo se comporta cada modelo frente a datos que no han visto con anterioridad.

La técnica *k-fold cross-validation* se ha empleado para el entrenamiento y validación. Esta técnica consiste en aleatorizar los datos y separarlos en *k* subconjuntos, donde cada uno de ellos tiene un tamaño similar. Un subconjunto es usado para validar el modelo que se ha entrenado a partir del resto de subconjuntos. Este proceso se repite *k* veces, de modo que cada subconjunto se usa una sola vez para validar. El error promedio entre todas las *k* particiones se reporta como Ɛ. En la *Figura 27* se muestra un ejemplo de k = 4.

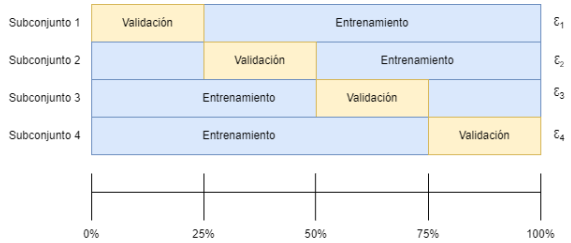


Figura 27. Ejemplo de cross-validation [Elaboración propia]

Típicamente, un valor de *k* bajo significa que el subconjunto de datos para el entrenamiento es bajo y para la validación es alto, en porcentaje. Esto podría dar lugar a que el error medio obtenido mediante el error de cada *k* caso sea alto. En contraste, un valor de *k* alto, es decir, un porcentaje más alto para el entrenamiento y uno bajo para la validación daría como resultado un error Ɛ de media más bajo. De normal para este tipo de técnicas es recomendable emplear un valor de *k* entre 5 y 20 [30]. Para este proyecto se ha decidido emplear un total de 5 subconjuntos para ambos tipos de cavidad. De esta forma, se tiene un total de 5 instancias de entrenamiento con un 80% de los datos para el entrenamiento y un 20% para la validación.

Los modelos de *Machine Learning* que se han seleccionado son aquellos que requieren un bajo coste computacional y de memoria, ya que los modelos que se elijan deberán de implementarse dentro de un microcontrolador. Los siguientes modelos fueron seleccionados: *coarse tree*, *medium tree*, *fine tree*, *linear discriminant*, *quadratic discriminant*, *logistic regression*, *linear SVM*, *quadratic SVM*, *cubic SVM*, *narrow neural*, *médium neural*, *wide neural network* y *bilayered neural*. Para el entrenamiento, validación y testeo se ha empleado la técnica "*wrapper*" [31]. Se selecciona entre dos y cuatro características, de las diez que mejor resultado han obtenido, y mediante estos se entrena y valida un modelo y después se testea, se obtiene como resultado una puntuación tanto del entrenamiento-validación como del testeo. Este proceso se repite varias veces combinando un grupo de características y seleccionando un modelo. Las métricas empleadas para determinar si un conjunto características-modelo presenta una buena puntuación son: exactitud, recuperación, precisión y *F1-score* [32]. Los modelos que se seleccionen serán un compromiso entre mejor resultado, bajo coste computacional y memoria y una selección de aquellas características que sean fáciles de obtener.

4.2.1 Resultados obtenidos del entrenamiento

En las *Tablas 5* y *6* se puede observar el resultado de emplear un *k* igual a 5. En estas tablas, se indica el número de muestras que hay para cada parte y de estas cuantas son correctas y cuantas muestran una fuga, y el porcentaje que representa respecto al total de muestras obtenidas. Se ha intentado aleatorizar los datos para que haya más datos con fuga en el testeo que

el *cross-validation*. De este modo, las condiciones de testeo a las que se deben de enfrentar los modelos serán más difíciles, ya que habrán tenido menos casos para determinar cuándo una señal presenta una fuga.

REC			
Subconjunto de datos	**Clase**	**Numero de muestras**	**Porcentaje**
Cross-Validation	Correcto	878	82,13%
	Fuga	191	17,87%
Test	Correcto	211	79,03%
	Fuga	56	20,97%
Total	Correcto	1089	81,51%
	Fuga	247	18,49%

Tabla 5. Porcentajes para cada uno de los subconjuntos de datos del REC [Elaboración propia]

AEC			
Subconjunto de datos	**Clase**	**Número de muestras**	**Porcentaje**
Cross-Validation	Correcto	922	86,20%
	Fuga	173	13,80%
Test	Correcto	216	81,62%
	Fuga	58	18,38%
Total	Correcto	1138	85,28%
	Fuga	231	14,72%

Tabla 6. Porcentajes para cada uno de los subconjuntos de datos del AEC [Elaboración propia]

Una vez que se han dividido los datos, se ha realizado el entrenamiento y testeo de varios modelos de Machine Learning. A continuación, se muestran las diez combinaciones modelo-características que mejor resultado han dado para cada tipo de cavidad.

En las *Tablas 7* y *8* se puede observar el resultado de las diez mejores combinaciones que se han podido realizar para las cavidades tipo REC.

Resultados obtenidos de los modelos predictos para la detección de fugas en los REC															
Algoritmo	**Características**	**5-folds Cross-Validation**						**Test**						**Total**	
		TP	**FP**	**TN**	**FN**	**Acc.**	**Recall**	**TP**	**FP**	**TN**	**FN**	**Acc.**	**Recall**	**Acc.**	**Recall**
Mediunm Tree	a2, a5, a6	875	4	187	3	99,35%	99,66%	209	2	54	2	98,50%	99,05%	99,18%	99,54%
Linear Discriminant	a2, a5, a7, a8, a17	872	20	171	6	97,57%	99,32%	205	6	50	6	95,51%	97,16%	97,16%	98,90%
Quadratic Discriminant	a2, a3, a4, a11	877	12	179	1	98,78%	99,89%	211	6	50	0	97,75%	100,00%	98,58%	99,91%
Logistic Regression	a2, a4	866	11	180	12	97,85%	98,63%	207	4	52	4	97,00%	98,10%	97,68%	98,53%
SVM Linear	a1, a2, a4, a7	868	7	184	10	98,41%	98,86%	206	6	50	5	95,88%	97,63%	97,90%	98,62%
SVM Quadratic	a2, a4, a7, a11	872	9	182	6	98,60%	99,32%	210	3	53	1	98,50%	99,53%	98,58%	99,36%
Narrow Neural Network	a2, a5, a11, a17	867	9	182	11	98,13%	98,75%	210	2	54	1	98,88%	99,53%	98,28%	98,90%
Medium Neural Network	a1, a2, a5, a7, a11, a17	875	7	184	3	99,06%	99,66%	208	1	55	3	98,50%	98,58%	98,95%	99,45%
Wide Neural Network	a1, a2, a5, a7, a17	870	7	184	8	98,60%	99,09%	211	1	55	0	99,63%	100,00%	98,80%	99,27%
Bilayered Neural Network	a1, a2, a3	872	5	186	6	98,97%	99,32%	209	4	52	2	97,75%	99,05%	98,73%	99,27%

Tabla 7. *Resultados obtenidos REC [Elaboración propia]*

Resultados obtenidos de los modelos predictos para la detección de fugas en los REC													
Algoritmo	Características	5-folds Cross-Validation				Test				Total			
		Acc.	Recall	Pre.	F1 Sc.	Acc.	Recall	Pre.	F1 Sc.	Acc.	Recall	Pre.	F1 Sc.
Mediunm Tree	a2, a5, a6	99,35	99,66	99,54	99,60	98,50	99,05	99	99,05	99,18	99,54	99,45	99,50
Linear Discriminant	a2, a5, a7, a8, a17	97,57	99,32	97,76	98,53	95,51	97,16	97	97,16	97,16	98,90	97,64	98,27
Quadratic Discriminant	a2, a3, a4 a11	98,78	99,89	98,65	99,26	97,75	100,00	97	98,60	98,58	99,91	98,37	99,13
Logistic Regression	a2, a4	97,85	98,63	98,75	98,69	97,00	98,10	98	98,10	97,68	98,53	98,62	98,58
SVM Linear	a1, a2, a4, a7	98,41	98,86	99,20	99,03	95,88	97,63	97	97,40	97,90	98,62	98,80	98,71
SVM Quadratic	a2, a4, a7, a11	98,60	99,32	98,98	99,15	98,50	99,53	99	99,06	98,58	99,36	98,90	99,13
Narrow Neural Network	a2, a5, a11, a17	98,13	98,75	98,97	98,86	98,88	99,53	99	99,29	98,28	98,90	98,99	98,94
Medium Neural Network	a1, a2, a5, a7, a11, a17	99,06	99,66	99,21	99,43	98,50	98,58	100	99,05	98,95	99,45	99,27	99,36
Wide Neural Network	a1, a2, a5, a7, a17	98,60	99,09	99,20	99,15	99,63	100,00	100	99,76	98,80	99,27	99,27	99,27
Bilayered Neural Network	a1, a2, a3	98,97	99,32	99,43	99,37	97,75	99,05	98	98,58	98,73	99,27	99,17	99,22

Tabla 8. Puntuaciones obtenidas en cada uno de los modelos del REC [Elaboración propia]

De entre todos estos modelos se ha decidió seleccionar el *logistic regression* debido a que emplea únicamente dos variables que son la media (a_2) y la diferencia entre el inicio y final del estado estacionario (a_4) y presenta unos valores de exactitud del 97,68%, 98,53% de exhaustividad, 98,62% de precisión y 98,58% de F1-score. Una alternativa a este modelo podría haber sido el *Medium Tree*, el cual presenta unos resultados de testeo mucho mejores, pero presenta la desventaja de emplear la desviación típica, a_6 (desviación típica de los datos que van desde el inicio de la recepción de datos hasta el flanco de subida de la electroválvula de llenado). El cálculo de esta característica es bastante elevado para el microcontrolador y es por ello por lo que no se ha seleccionado este modelo.

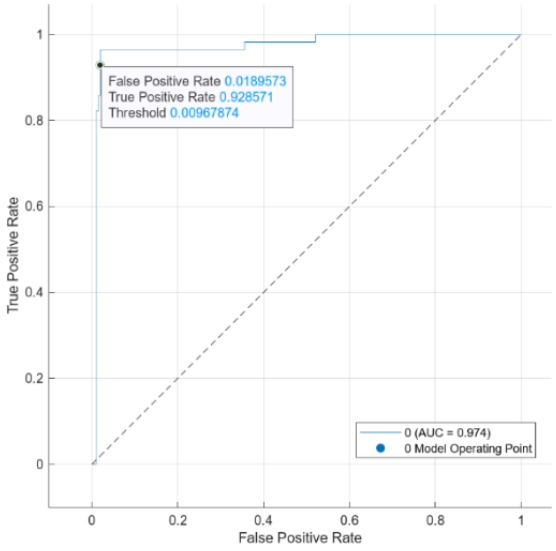


Figura 28. Representación ROC del REC

En la *Figura 28* se puede observar la curva ROC del modelo *logistic regression*. Este tipo de gráficas se obtiene a partir de varios umbrales (un umbral indica si el resultado de introducir un dato en un modelo se considera positivo o falso). A partir de estos umbrales, que van de 0 a 1, se va trazando la curva. El "*Model Operating Point*" que se observa en la *Figura 28* corresponde al

umbral seleccionado por *Matlab* para el modelo seleccionado. Otra medida adicional agregada al rendimiento de cada uno de los umbrales es el AUC, que nos indica cómo se comporta el modelo en términos generales, en este caso se ha conseguido un valor del 0,974 esto significa que el modelo es capaz de acertar el 97,4% de las veces.

En el caso de los AEC, las diez mejores combinaciones que se han podido obtener se pueden observar en las *Tablas 9* y *10*.

Resultados obtenidos de los modelos predictos para la detección de fugas en los AEC															
Algoritmo	Características	5-folds Cross-Validation						Test						Total	
		TP	FP	TN	FN	Acc.	Spe.	TP	FP	TN	FN	Acc.	Spe.	Acc.	Spe.
Fine tree	b1, b2	922	0	173	0	100%	100%	216	0	58	0	100%	100%	100%	100%
	b1, b2, b5	919	3	170	3	99%	98%	216	0	58	0	100%	100%	100%	99%
Coarse tree	b2, b8	921	3	170	1	100%	98%	216	0	58	0	100%	100%	100%	99%
Linear Discriminant	b2, b4,b8, b10	918	8	165	4	99%	95%	215	1	57	1	99%	98%	99%	96%
Quadratic Discriminant	b2, b4, b7, b11	910	6	167	12	98%	97%	212	1	57	4	98%	98%	98%	97%
Logistic Regression	b4, b8, b11	920	3	170	2	100%	98%	216	0	58	0	100%	100%	100%	99%
Linear SVM	b1, b4, b8	921	2	171	1	100%	99%	216	0	58	0	100%	100%	100%	99%
Quadratic SVM	b4, b8, b11	920	3	170	2	100%	98%	216	0	58	0	100%	100%	100%	99%
Medium Neural Network	b4, b8	918	3	170	4	99%	98%	216	0	58	0	100%	100%	99%	99%
Wide Neural Network	b2, b8	917	3	170	5	99%	98%	213	0	58	3	99%	100%	99%	99%

Tabla 9. Resultados obtenidos AEC [Elaboración propia]

Resultados obtenidos de los modelos predictos para la detección de fugas en los AEC													
Algoritmo	Características	5-folds Cross-Validation				Test				Total			
		Acc.	Recall	Pre.	F1 Sc.	Acc.	Recall	Pre.	F1 Sc.	Acc.	Recall	Pre.	F1 Sc.
Fine tree	b1, b2	100,00	100,00	100,00	100,00	100,00	100,00	100	100,00	100,00	100,00	100,00	100,00
	b1, b2, b5	99,45	99,67	99,67	99,67	100,00	100,00	100	100,00	99,56	99,74	99,74	99,74
Coarse tree	b2, b8	99,63	99,89	99,68	99,78	100,00	100,00	100	100,00	99,71	99,91	99,74	99,82
Linear Discriminant	b2, b4,b8, b10	98,90	99,57	99,14	99,35	99,27	99,54	100	99,54	98,98	99,56	99,21	99,39
Quadratic Discriminant	b2, b4, b7, b11	98,36	98,70	99,34	99,02	98,18	98,15	100	98,83	98,32	98,59	99,38	98,99
Logistic Regression	b4, b8, b11	99,54	99,78	99,67	99,73	100,00	100,00	100	100,00	99,63	99,82	99,74	99,78
Linear SVM	b1, b4, b8	99,73	99,89	99,78	99,84	100,00	100,00	100	100,00	99,78	99,91	99,82	99,87
Quadratic SVM	b4, b8, b11	99,54	99,78	99,67	99,73	100,00	100,00	100	100,00	99,63	99,82	99,74	99,78
Medium Neural Network	b4, b8	99,36	99,57	99,67	99,62	100,00	100,00	100	100,00	99,49	99,65	99,74	99,69
Wide Neural Network	b2, b8	99,27	99,46	99,67	99,57	98,91	98,61	100	99,30	99,20	99,30	99,74	99,52

Tabla 10. Puntuaciones obtenidas en cada uno de los modelos del AEC [Elaboración propia]

De los modelos obtenidos para el AEC se ha seleccionado el *Coarse Tree*. Esto se debe por su sencillez, ya que es el modelo más sencillo de implementar y emplea solamente dos características fáciles de calcular, ya que b_2 es la media de presión en el régimen estacionario y el b_8 es la diferencia de presión entre el inicio de la recepción de datos y el flanco de subida de la electroválvula de llenado. Presenta unos valores de exactitud del 99,71%, 99,91% de exhaustividad, 99,74% de precisión y 99,82% de F1-score, los cuales son superados únicamente por el *fine tree*, pero este presenta una mayor complejidad que el *coarse tree*, aunque los dos son de los más sencillos de implementar.

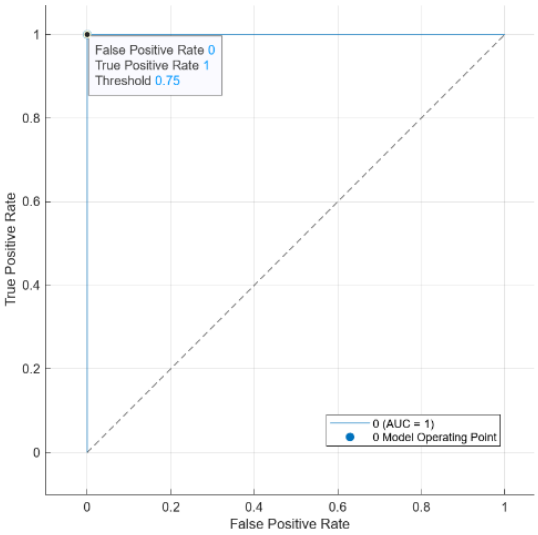


Figura 29. Representación ROC del AEC

En la *Figura 29* se puede observar el ROC del AEC. En este caso a diferencia del REC, se tiene una curva de ROC recta, que hace un ángulo de 90 grados, esto significa que el modelo es capaz de acertar el 100% de los casos. Esta característica también viene indicada por el AUC el cual es igual a 1.

4.3 Implementación en el microcontrolador

Una vez que se tienen los modelos entrenados y validados en *Matlab*, el siguiente paso consiste en exportar el modelo a código C, para poder llegar a integrarlo junto al resto del código del sistema Endoworm. Es importante mencionar que esta parte del TFM se va a limitar a exportar y validar el modelo en una placa de STM32, la integración con el resto del código de control se llevará a cabo más adelante, escapándose del alcance del presente TFM.

4.3.1 Obtención del modelo

Para exportar el modelo se va a hacer uso de la herramienta *Matlab Coder*, la cual nos permite convertir una función cualquiera de *Matlab*, a código C. Si abrimos *Matlab Coder* e importamos la función que nos interesa convertir, *Matlab* nos preguntará por los argumentos de entrada, los cuales se tendrán que seleccionar de forma manual. Sin embargo, si en vez de importar nuestra función importamos un fichero en el que únicamente se hace una llamada a nuestra función, *Matlab* detectará automáticamente los argumentos de entrada. Una vez que tenemos los

argumentos, el siguiente paso consiste en realizar una serie de comprobaciones para determinar si es posible generar el código. Finalmente, seleccionamos que tipo de fichero queremos, si es un código fuente, una libraría dinámica o estática o un ejecutable, en este caso seleccionamos código fuente y además que este en C y no C++.

Una vez que la conversión ha terminado *Matlab* generará varios módulos: un "*main.c*" donde se pondrá un ejemplo de cómo usar el modelo generado, otro fichero con el modelo y una serie de librerías donde están definidos unas macros y tipos de variables (esta librería es común para todos los modelos de *Machine Learning*, por lo que solo es necesario importarlo una vez).

4.3.2 Buffer y cálculo de características

El modelo generado mediante *Matlab Coder* presenta unos argumentos de entrada, los cuales son las características que hemos empleado para cada modelo de *Machine Learning*. Estos argumentos deben ser previamente calculados, para ello se necesita una función que vaya recopilando los datos de las señales, y a partir de estos se puedan calcular las características y normalizarlas (los modelos de *Machine Learning* emplean características normalizadas, tal como se mencionó en el apartado 4.1).

Para almacenar los datos de las señales se ha decidido emplear un buffer circular. Este buffer comenzará a recopilar datos cuando se alcance la muestra número 100, antes de que llegue el flanco de subida de la electroválvula de llenado, hasta llegar a la muestra número 100, después de detectar el flanco de bajada de la electroválvula de vaciado. Los modelos que se han seleccionado para el AEC y REC no se necesitan las últimas 100 muestras, pero se ha decidido mantenerlas por si en el futuro se necesitarán.

El motivo por el cual se necesita un buffer circular se puede explicar a partir de la *Figura 30*. Tal como se observa cuando el sistema Endoworm se enciende, los datos que le llegan al buffer no son útiles, por lo que se tienen que ir escribiendo y borrando hasta llegar a la muestra de interés. Por este motivo se eligió un buffer circular en lugar de uno lineal ya que esta tarea se puede realizar fácilmente, sobrescribiendo los datos que había por lo nuevos y detenerse cuando es necesario. En el Anexo A2.1 se puede ver como se implementó este buffer circular.

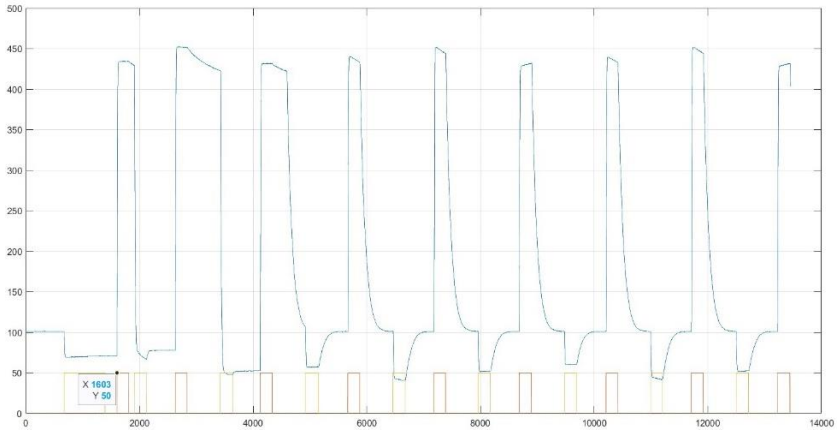


Figura 30. Ejemplo de señal AEC

En la *Figura 31* se puede observar un diagrama funcional de como se ha implementado el buffer circular.

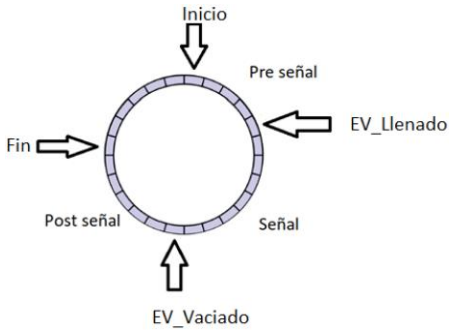


Figura 31. Representación del buffer circular

Tal como se observa, el buffer está separado en tres partes: *Pre señal*, encargado de almacenar las primeras 100 muestras, en esta parte del buffer los datos se van sobrescribiendo hasta que se detecta el flanco de subida de la electroválvula de llenado, momento en que el apuntador del buffer pasa a apuntar a la muestra número 100 y se empieza a almacenar los datos hasta llegar al flanco de subida de la electroválvula de vaciado, esta parte se denomina *Señal*, y finalmente el *Post señal* donde se almacena los 100 últimos datos después de detectar el flanco de bajada de la electroválvula de vaciado.

Un factor importante a la hora de diseñar el buffer es el tamaño que tendrá, ya que las señales que le llegarán tendrán un número de muestras distintos, sin embargo, el tamaño de un buffer debe estar definido y no puede alterarse en función de la longitud de la señal. Para determinar el tamaño se tuvo en cuenta el tamaño de todas las señales segmentadas que se emplearon para el entrenamiento y testeo de modelo de Machine Learning y se escogió aquella señal que tuviera un número de muestras mayor y se le añadió un 20% más, para poder cubrir cualquier señal que pudiese llegar al buffer. En caso de que no se llegue a completar el buffer no habrá ningún problema ya que a medida que el buffer esta almacenando datos también existen variables que van apuntando los instantes donde existen flancos y cuando empieza y termina la señal por lo que en todo momento se sabe la longitud real del buffer.

Una vez que se han obtenido todas las muestras de la señal segmentada, se calculan las características y se normalizan empleando el método z-score. Tal como se indicó en el apartado 4.1, los valores de media y desviación típica empleados para la normalización de las características provienen del entrenamiento de los modelos en *Matlab*, por lo que ya están previamente calculados.

Finalmente, estas características normalizadas se pasan como argumentos a las funciones de Machine Learning generadas en *Matlab* y se obtiene como salida una etiqueta que indica si la señal que se ha analizado presenta o no una fuga.

4.3.3 Modelos generados

Tal como se indicó en el apartado 4.2.1, el modelo obtenido para el AEC es el *coarse tree* y para el REC el "*logistic regression*. En este apartado se va a comentar más en detalle que son estos modelos y como son capaces de distinguir, a partir de una señal de presión, si una cavidad presenta o no una fuga.

Logistic regression:

Este tipo de modelo forma parte del grupo de aprendizaje supervisado de Machine Learning y es principalmente empleado para predecir la probabilidad de que una instancia pertenezca a una clase dada.

Este tipo de modelo se parece bastante a la *linear regression*, pero se diferencian por el uso que se le dan. La *linear regression* se emplea para solucionar problemas de regresión (por ejemplo, predecir que una persona pese Y conociendo su altura X) mientras que la *logistic regression* se emplea para problemas de clasificación, y por lo tanto emplea variables dicotómicas como salida o resultado (la variable Y solo puede tomar dos valores que representan la ocurrencia o ausencia de un determinado suceso). Es por ello por lo que se ha utilizado la *logistic regression* para el entrenamiento de las señales segmentadas REC ya que lo importante es definir si una señal presenta o no una fuga. Se ha definido que una señal no presenta fuga como 0 y 1 cuando si que tiene fuga. Luego el valor que se va a obtener va a variar entre 0 y 1, un valor cercano a 0 implicará que la señal presenta una fuga mientras que un valor próximo a 1 implicará que la señal no presenta una fuga.

La forma característica que presenta la función *logistic regression* es una sigma, con valores muy pequeños cuando la entrada 'X', tiene valores cercanos a 0 y va creciendo exponencialmente hasta llegar a acercarse a 1, momento en el que comienza a estabilizarse, tal como se observa en la *Figura 32*.

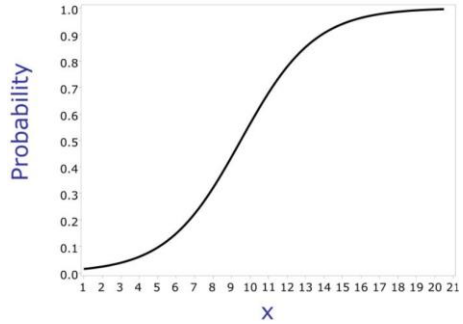


Figura 32. Representación gráfica de logistic regression

A la función de *logistic regression* se le tiene que imponer dos condiciones: forzar a que los valores predichos de probabilidad se ubiquen entre 0 y 1, independientemente de los valores que tome la variable de entrada, X y que se verifique la relación sigmoidal que se observa en la *Figura 32*.

Estas dos condiciones se consiguen mediante la transformación logística, que consiste en aplicar el logaritmo al *Odds*, esto es la probabilidad de que un suceso ocurra dividido por la probabilidad de que no ocurra, dado un valor de la variable X.

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X \tag{3}$$

Donde β_0 y β_1 son los coeficientes de regresión y X la variable de entrada. Esta ecuación (3) también se puede definir como:

$$\frac{p}{1-p} = e^{\beta_0+\beta_1 x} \tag{4}$$

Despejando la probabilidad p en (4) se obtiene la probabilidad de que ocurra el suceso:

$$p = \frac{1}{1 + e^{-\beta_0-\beta_1 x}} \tag{5}$$

Mediante esta ecuación (4) determinamos la probabilidad de que un evento ocurra en función de una entrada X. Los valores de coeficientes de regresión se pueden obtener empleando técnicas como el algoritmo de Descenso de Gradiente [33]. Cabe destacar que si se emplea más de una entrada se puede ajustar esta ecuación añadiendo tantos coeficientes β_n como valores de entrada se tengan X_n.

A partir de los coeficientes que se han obtenido al entrenar el modelo en *Matlab* se ha podido obtener la fórmula (6), del modelo de *logistic regression*.

$$p = \frac{1}{1 + e^{-(4.692 + 4.034x_1 + 2.215x_2)}} \quad (6)$$

Siendo la variable x_1 la media en el estado estacionario (a_2) y x_2 la diferencia entre el punto de inicio y final del estado estacionario (a_4).

Coarse tree:

El *Coarse tre*e pertenece al grupo de árbol de decisiones, el cual se considera uno de los preferidos para solucionar problemas de clasificación debido a su sencillez. Tienen unas entradas que pueden ser un objeto o una situación descrita mediante un conjunto de atributos y a partir de estos se obtiene una respuesta.

En general los árboles de decisión constan de una serie de nodos que forman un árbol enraizado, entre estos se encuentra el nodo raíz, el cual es el es punto de partida y se expande hacia el resto de ramas dando la forma característica de un árbol. Las ramas salientes del nodo raíz alimentan los nodos internos denominados nodos de decisión. Ambos nodos se encargan de realizar evaluaciones y de este modo determinar cuál es el siguiente nodo a evaluar hasta llegar a obtener una conclusión. Cada nodo de decisión puede llegar a tener varias salidas en función de los atributos de entrada. El caso más simple y común es cada nodo de decisión presente un solo atributo y a partir de este se consideren las salidas que pueda tener el nodo. Una vez que se ha navegado a lo largo de los nodos de decisión se llega, finalmente, a los nodos hoja, los cuales representan los resultados posibles.

Dentro de los árboles de decisión existen tres tipos: *Fine tree*, *Medium tree* y *Coarse tree*. Estos se diferencian principalmente por el número de divisiones que se pueden llegar a hacer. *Fine tree* presenta un número máximo de 100 divisiones por lo que se pueden llegar a hacer distinciones detalladas entre clases, *Medium tree* tiene un máximo de 20 divisiones por lo que puede distinguir bastante bien entre clases y finalmente el *Coarse tree* el cual solo pueden hacer unas 4 divisiones por lo que solo es capaz de realizar una distinción amplia entre clases, sin llegar al detalle [34]. Esto indica que el modelo seleccionado a pesar de ser el que menos divisiones y por ende el más sencillo de implementar, es capaz de acertar con una exactitud del 99.71% y una precisión del 99.74%.

En la *Figura 33* se muestra el modelo de Coarse Tree empleado para las cavidades tipo AEC.

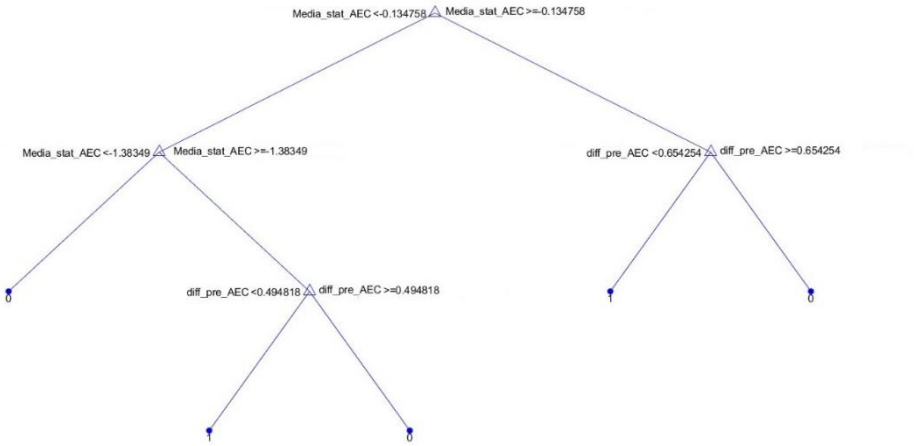


Figura 33. Representación del modelo Coarse Tree [Elaboración propia]

4.3.4 Estimación de tiempos y tamaño de funciones

Un parámetro que se debe tener en cuenta a la hora de desarrollar código que se va a integrar en un proyecto más grande, son los tiempos de ejecución que necesita cada función. Las funciones que se han implementado son tres: la primera función se encarga de almacenar los datos en un buffer circular (por lo tanto, será aquella que más tiempo necesite), una segunda encargada de calcular las características y normalizarlas y finalmente la función que determina si una señal tiene o no una fuga. Son tres funciones y cada una de ella se tiene que ejecutar cuatro veces (una por cada cavidad, recordemos que las cavidades son: DAEC, PAEC, MREC y FREC), por lo que para determinar la presencia o ausencia de una fuga de aire en todas las cavidades se necesitarán un total doce ejecuciones. Por este motivo es importante estimar los tiempos que consumen cada una de estas.

Para determinar el tiempo que tarda cada función en ejecutarse se ha empleado un contador de ciclos (DWT) que dispone el microcontrolador de STM32. Este mide el número de ciclos necesarios para ejecutar una línea de código, y sabiendo la frecuencia de reloj se puede determinar el tiempo que tarda una función en ejecutarse, por ejemplo, si se necesitan un total de 120 ciclos y el reloj funciona a 16MHz entonces la función ha tardado 7,5 microsegundos. El cálculo de tiempos se ha basado en dos registros a partir de los cuales se ha obtenido un total de veinte ocho señales segmentadas, sabiendo que la frecuencia de reloj del sistema Endoworm es de 168MHz, se ha calculado la media y desviación típica, los resultados se pueden observar en las *Tabla 11* y *Tabla 12*.

	AEC		
	Buffer	**Características**	**Resultados**
Tiempo medio (µs)	1.956,79	539,40	19,48
Desviación típica del tiempo medio (µs)	280,05	73,54	0,41

Tabla 11. Estimación de tiempos de las señales AEC [Elaboración propia]

	REC		
	Buffer	**Características**	**Resultados**
Tiempo medio (µs)	2.025,41	587,90	30,04
Desviación típica del tiempo medio (µs)	297,40	110,98	0,12

Tabla 12. Estimación de tiempos de las señales REC [Elaboración propia]

Tal como se observa en las *Tabla 11* y *Tabla 12*, la función que más tarda en ejecutarse es la de almacenar los datos en los buffers, ya que esta función se ejecuta continuamente y no termina hasta que se guarda el último dato, se ha estimado también el tiempo que tarda en realizarse una sola ejecución y es de 1,1845 microsegundos. En segundo lugar, se encuentra el cálculo de las características, la razón por la que necesita tanto tiempo se debe a que se calcula la media en el régimen estacionario, para lo cual se necesitan varias iteraciones, en caso de que este tiempo sea demasiado alto se puede realizar la suma de datos en el buffer y únicamente realizar una división en esta función. Finalmente, la función que menos tiempo tarda es la que determina la presencia de una fuga en las señales, si se llegase a necesitar bajar los tiempos de ejecución se podría llegar a plantear en sustituir las operaciones de división para que se realizasen en punto fijo en lugar de punto flotante, ya que *Matlab* permite generar este tipo de operaciones.

A parte de estas opciones también se pueden plantear otro tipo de estrategias para conseguir disminuir los tiempos de ejecución como por ejemplo añadir la característica *"inline"* a las funciones, de este modo en vez de realizarse llamadas a las funciones se realizarán copias en los puntos donde se haga la llamada a la función, con esto se consigue reducir los tiempos de ejecución, pero en contraste se aumenta el tamaño del código.

Otro factor interesante es el tamaño que tienen las funciones y variables que se emplean. Para ello se ha empleado el "*Build Analyzer*", que lleva incorporado *STM32CubdeIde*.

En el caso de las variables, se han utilizado un total de 12 variables para ambos tipos de modelos, los cuales ocupan un total de 37,76Kb (estos cálculos se han hecho teniendo en cuenta solo un buffer para cada tipo de cavidad, en el caso de que fueran dos buffers para cada cavidad se tendría que multiplicar por 2). Por otro lado, las funciones que se han desarrollado no ocupan mucha memoria, con un total de 1,786Kb.

En total, teniendo en cuenta las variables y funciones se tiene un consumo de 39,594Kb de memoria, lo cual no se considera mucho. Sin embargo, si el proyecto se integra junto al resto del sistema Endoworm 4.0 se debe de analizar de nuevo y evaluar si puede haber problemas de almacenamiento y en caso hubiera considerar extender la memoria RAM.

Name	Run address (V...	Load address (LM...	Size
RAM	0x20000000		192 KB
.data	0x20000000	0x08003798	38,92 KB
buffer	0x20000000	0x08003798	6,68 KB
buffer1	0x20001abc	0x08005254	6,68 KB
Press_DAEC	0x20003578	0x08006d10	8,14 KB
Fill_valve_DAEC	0x20005604	0x08008d9c	2,03 KB
Vacc_valve_DAEC	0x20005e28	0x080095c0	2,03 KB
Press_FREC	0x2000664c	0x08009de4	8,84 KB
Fill_valve_FREC	0x200089a4	0x0800c13c	2,21 KB
Vacc_valve_FREC	0x2000927c	0x0800ca14	2,21 KB
SystemCoreClock	0x20009b54	0x0800d2ec	4 B
uwTickPrio	0x20009b58	0x0800d2f0	4 B
uwTickFreq	0x20009b5c	0x0800d2f4	1 B
_impure_data	0x20009b60	0x0800d2f8	76 B
_impure_ptr	0x20009bac	0x0800d344	4 B
.bss	0x20009bb0		404 B
timerdelay	0x20009bcc		12 B
dv	0x20009bd8		16 B
dv2	0x20009be8		16 B
dv1	0x20009bf8		8 B
label	0x20009c00		8 B

(a)

Name	Run address (V...	Load address (LM...	Size
main	0x08000e14	0x08000e14	216 B
write	0x08000eec	0x08000eec	88 B
read	0x08000f44	0x08000f44	76 B
readfromsensors	0x08000f90	0x08000f90	602 B
calculateAECparams	0x080011ec	0x080011ec	440 B
calculateRECparams	0x080013a4	0x080013a4	444 B
resetbuffer	0x08001560	0x08001560	140 B
SystemClock_Config	0x080015ec	0x080015ec	212 B
MX_GPIO_Init	0x080016c0	0x080016c0	1,21 KB
Error_Handler	0x08001b98	0x08001b98	10 B
HAL_MspInit	0x08001ba4	0x08001ba4	80 B

(b)

Figura 34. Espacio que ocupan las variables (a) y las funciones (b). [Elaboración propia].

4.3.5 Verificación de los modelos

Una vez obtenidos los modelos de Machine Learning, el último paso consiste en verificar que estos modelos funcionan igual que los obtenidos en *Matlab*. Para ello se ha empleado la placa de desarrollo *32F429IDISCOVERY*, la cual usa el mismo microcontrolador que la placa de control del sistema Endoworm 4.0. Mediante esta placa y el software de programación *STM32CubeIde* se ha creado un proyecto nuevo. En este proyecto se han importado los modelos de Machine Learning y se han inicializado seis buffers, dos de ellos para guardar las señales de presión de las cavidades AEC y REC y las otras cuatro para guardar los pulsos generados por las electroválvulas de llenado y vaciado, de cada cavidad. A partir de estos buffers se obtienen y se normalizan las características que necesita cada modelo. Finalmente, estas características son enviadas a los modelos, para que a partir de estos sean capaces de evaluar, si en las señales de presión que se han almacenado en los buffers existe alguna con fuga (recordemos que estos modelos son simplemente funciones que aceptan unos parámetros de entrada y devuelven un valor).

Después de crear el proyecto, se almacenan en estos seis buffers las muestras obtenidas en 2 registros, seleccionados de forma aleatoria. Dentro de estos registros, existen un total de 24 señales, las cuales son demasiadas para introducirlas al mismo tiempo, por lo que se ha decidido

separar la validación en 3 etapas, cargando solo 8 señales de cada cavidad en la placa de desarrollo. En paralelo, a través de la herramienta de *Matlab*, se calculan las características, que necesita cada modelo, de estos 2 registros y se introducen como entradas a la función que se ha generado al exportar los modelos desde la App *Classification Learner*, a partir de estas funciones se obtiene un resultado que es el estado de las señales. Para poder visualizar el cálculo de las características como el estado de las cavidades se ha empleado la herramienta de depuración interna de *STM32CubeIde*. Tanto los resultados obtenidos a las salidas de los modelos de clasificación predictiva como las características calculadas han sido comparadas con los resultados obtenidos mediante *Matlab*, dando idénticos resultados. Por lo que se puede concluir diciendo que el comportamiento de los modelos predictivos de clasificación que se han implementado en el microcontrolador se comporta prácticamente igual que los de *Matlab*. Esto quiere decir que los desempeños de estos modelos en el microcontrolador serán iguales a los que se han obtenido y calculado en *Matlab*.

Capítulo 5. Conclusiones

Los objetivos del presente proyecto se han centrado en la implementación de una interfaz gráfica que fuera más intuitiva y amigable con el usuario y el desarrollo de un sistema de protección en caso de que alguna de las cavidades presentase alguna fuga afectando a la seguridad del paciente y al funcionamiento normal del dispositivo.

Respecto a la nueva interfaz gráfica al cambiar de modelo, por uno de la marca *Emerging Display Technologies CORP*, el desarrollo de esta se ha tenido que realizar desde el principio, introduciendo algunas mejoras, pero manteniendo las funciones que había en la versión anterior. Se ha modificado la ventana principal para que este menos cargada de botones, permitiendo que se aprecie mejor el estado de las cavidades. Respecto a las cavidades, se han hecho una serie de cambios, una de ellas es que ahora desde el punto de vista del usuario solo existe una sola cavidad de expansión axial en lugar de dos. Esto se debe a que los AEC actúan de forma antagonista, por lo que se pueden configurar ambos con los mismos parámetros. Por otro lado, para que sea más intuitivo y fácil de recordar, se han cambiado sus respectivos nombres, en lugar de llamarse FREC, MREC, DAEC y PAEC se llamarán balón fijo, balón móvil y fuelles, respectivamente. Además, se han reducido el número de pulsaciones para navegar a lo largo de las pantallas y se ha limitado cada pantalla a una función específica, por lo que ahora son más entendibles. Gracias a todas estas mejoras, se ha conseguido aumentar la usabilidad de la pantalla.

Por otro lado, se ha mejorado el sistema de comunicaciones entre la pantalla y la placa de control. En la pantalla anterior las tramas que se enviaban eran fijas y solo permitían enviar datos en formato ASCII, mientras que ahora se codifican en decimal, optimizando de este modo el número de datos a enviar y haciendo las tramas de comunicación más sencillas.

Respecto a la pantalla se puede concluir que todos los cambios que se han hecho, tanto internos como externos, han supuesto una mejora sustancial respecto al modelo anterior. Se ha mejorado la experiencia del usuario y gracias a la optimización de las tramas se ha conseguido reducir el tiempo que tarda la placa de control en procesarlas.

En cuanto al sistema de detección de fallo, se han obtenido resultados prometedores con los algoritmos de Machine Learning empleados. Para las cavidades de expansión axial se ha empleado el modelo de "*coarse tree*" y las características b_2 y b_8, obteniendo una precisión del *99,74%*, y en el caso de las cavidades de expansión radial se ha seleccionado el modelo de "*logistic regression*" y las características a_2 y a_4, con una precisión del *98,62%*. Cabe mencionar que ambos modelos son de los más fáciles de implementar, siendo de estos dos el "*coarse tree*" el más fácil. La ventaja de estos modelos es que no presentan unos tiempos de ejecución elevados, en el apartado 4.3.4 se mostró sus tiempos de ejecución, el "*coarse tree*" se ejecuta en 19,48µs mientras que el "*logistic regression*" en 30,04µs. Por lo tanto, su integración con el resto del proyecto no debería ser muy costosa, la parte que puede llegar a dar problemas de sincronización es la recepción de datos ya que tarda unos milisegundos en llenarse los buffers. En principio no debería de haber problemas de integración mientras que los tiempos de ejecución de los modelos de clasificación sea inferior a 4ms. Esto se debe a que, en la placa de control, la lectura de los datos se realiza cada 5ms a través de la DMA y existe una máquina de estados que se ejecuta cada 5ms, siendo esta la que toma las decisiones importantes en el dispositivo en tiempo real. Pero su tiempo de ejecución es inferior a 300µs, por lo que, si después de llenar el buffer se consigue un resultado del estado de las cavidades en menos de 4ms, para cada cavidad, entonces no habría problemas para la integración del sistema de detección de fallos. Esto permite que todas las tareas se ejecuten en tiempo real sin la necesidad de recurrir a un RTOS (Real Time Operative System), que facilite la planificación de la ejecución de las tareas y permitiendo conservar la programación de microcontrolador de la placa de control del Endoworm 4.0 en baremetal.

Se llegaron a plantear otras técnicas de Machine Learning que tuviera unos resultados ligeramente superiores, sin embargo, son modelos muchos más complejos que darían como consecuencia una acción más lenta. También se valoraron otras herramientas, como por ejemplo el software

"*NanoEdge AI Studio*" de la marca de STM32, sin embargo, se descartó esta opción debido a que solo genera un modelo de Machine Learning con el que se podía interactuar, por lo tanto, se descartó esta opción debido a la imposibilidad de importar el modelo a otra marca de microcontroladores.

En el estado actual del proyecto, los dos modelos que se han seleccionado son de los que mejor resultado presentan y de los más sencillos de implementar, por lo que son la mejor opción. Sin embargo, se tendrán que evaluar con datos completamente nuevos para ver si siguen respondiendo igual de bien.

Capítulo 6. Futuras líneas de trabajo

En este apartado se numeran las posibles líneas de trabajo futuras que se podrían llegar a plantear para avanzar u optimizar el código que actualmente se ha desarrollado.

Respecto a la interfaz gráfica se han implementado todas las pantallas a excepción de la pantalla de ajustes, desde esta el usuario puede cambiar la hora, fecha, idioma y brillo de la pantalla, sin embargo, de todas estas opciones la única que se ha implementado es la del brillo de la pantalla. Por otro lado, la pantalla de diagnóstico es capaz de recibir y enviar órdenes a la placa control, sin embargo, en el estado actual del proyecto las instrucciones que la placa de control debe de ejecutar para diagnosticar las cavidades todavía no han sido implementadas. Esto se debe a que primero se han de integrar los modelos de clasificación de detección de fallos para terminar de concluir los últimos ajustes de la pantalla, escapándose al alcance de este TFM y dejándose para futuros trabajos.

Por otro lado, el medio de comunicación que se ha decidido emplear es el de RS232 por la seguridad que aporta. Aun así, en el futuro se podría llegar a plantear en cambiar este por el USB ya que la pantalla gráfica permite esta opción, esto será objeto de discusión.

Respecto a los modelos de Machine Learning, conviene mencionar que a pesar de que ambos modelos responden correctamente en la mayoría de los casos, cabe destacar que han sido entrenados empleando datos experimentales en "*ex-vivo*", es decir que proceden de intestinos artificiales o en ausencia de ellos, por lo que convendría evaluar su comportamiento en "*in-vivo*", es decir, en animales. En caso de que respondan correctamente se podría plantear en emplearse en el ámbito clínico. En contra, si los modelos no son capaces de evaluar las cavidades correctamente, convendría entrenar de nuevo los modelos con los datos que se obtengan en "*in-vivo*". De este modo ambos modelos tendrían una mayor cantidad y variabilidad de datos por lo que serían capaces de responder mejor. Si a pesar de entrenar de los modelos con datos nuevos no se consiguen mejoras se tendría que evaluar el uso de modelos más sofisticados como pueden ser las redes neuronales.

Capítulo 7. Bibliografía

[1] J. Tobella, V. Pons-Beltrán, A. Santonja, C. Sánchez, A. J. Campillo-Fernández, y A. Vidaurre, «Analysis of the 'Endoworm' prototype's ability to grip the bowel in in vitro and ex vivo models», *Proc Inst Mech Eng H*, vol. 234, n.º 5, pp. 468-477, may 2020, doi: 10.1177/0954411920901414.

[2] A. B. Slat, J. Burdick, y W. Grundfest, «The Development of a Robotic Endoscope», 1995. doi: 10.1109/IROS.1995.526155.

[3] C. Sánchez-Diaz, E. Senent-Cardona, V. Pons-Beltran, A. Santonja-Gimeno, y A. Vidaurre, «Endoworm: A new semi-autonomous enteroscopy device», *Proc Inst Mech Eng H*, vol. 232, n.º 11, pp. 1137-1143, nov. 2018, doi: 10.1177/0954411918806330.

[4] J. L. Rodriguez, «Como funciona». https://como-funciona.co/un-endoscopio/ (accedido 22 de mayo de 2023).

[5] Wikipedia contributors, «Enteroscopy - Wikipedia». https://en.wikipedia.org/wiki/Enteroscopy (accedido 10 de abril de 2023).

[6] M. A. Khashab *et al.*, «The role of deep enteroscopy in the management of small-bowel disorders», *Gastrointest Endosc*, vol. 82, n.º 4, pp. 600-607, oct. 2015, doi: 10.1016/j.gie.2015.06.046.

[7] S. KENJI y G. TEIICHI, «Fujifilm». https://www.fujifilm.com/sg/en/healthcare/endoscopy/endoscopy-scopes (accedido 19 de abril de 2023).

[8] Grastrohealthpartners, «Single Ballon Enteroscopy: A closer Look». https://www.gastrohealthpartners.com/single-balloon-enteroscopy-a-closer-look/ (accedido 5 de mayo de 2023).

[9] K. Kobayashi y T. Kucharzik, «Single-Balloon Enteroscopy», en *Video Capsule Endoscopy: A Reference Guide and Atlas*, M. Keuchel, F. Hagenmüller, y H. Tajiri, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 119-126. doi: 10.1007/978-3-662-44062-9_14.

[10] P. Esteban Delgado, «Factores Asociados a Complicaciones en Enteroscopia de Doble Balón en una Unidad de Referencia. Datos para la Seguridad del Paciente.», Murcia, nov. 2015.

[11] S. S. Chauhan *et al.*, «Enteroscopy», *Gastrointest Endosc*, vol. 82, n.º 6, pp. 975-990, dic. 2015, doi: 10.1016/j.gie.2015.06.012.

[12] A. May, L. Nachbar, M. Schneider, M. Neumann, y C. Ell, «Push-and-pull enteroscopy using the double-balloon technique: method of assessing depth of insertion and training of the enteroscopy technique using the Erlangen Endo-Trainer», *Endoscopy*, vol. 37, n.º 1, pp. 66-70, 2005, doi: 10.1055/s-2004-826177.

[13] X.-B. Li *et al.*, «A novel modality for the estimation of the enteroscope insertion depth during double-balloon enteroscopy», *Gastrointest Endosc*, vol. 72, n.º 5, pp. 999-1005, 2010, doi: 10.1016/j.gie.2010.07.045.

[14] S. Mehdizadeh *et al.*, «What is the learning curve associated with double-balloon enteroscopy? Technical details and early experience in 6 U.S. tertiary care centers», *Gastrointest Endosc*, vol. 64, n.º 5, pp. 740-750, nov. 2006, doi: 10.1016/j.gie.2006.05.022.

[15] A. Chacko, A. Kumar Dutta, K. Gangadharan Sajith, A. J. Joseph, y E. George Simon, «Learning curve, diagnostic yield and safety of single balloon enteroscopy», 2012.

[16] J. M. Buscaglia, «The spiral enteroscopy training initiative: results of a prospective study evaluating the Discovery SB overtube device during small bowel enteroscopy (with video)», *Thieme*, pp. 194-199, mar. 2009.

[17] C. Sánchez-Diaz, E. Senent-Cardona, V. Pons-Beltran, A. Santonja-Gimeno, y A. Vidaurre, «Endoworm: A new semi-autonomous enteroscopy device», *Proc Inst Mech Eng H*, vol. 232, n.º 11, pp. 1137-1143, nov. 2018, doi: 10.1177/0954411918806330.

[18] Roberto, «IMPLEMENTACIÓN DEL HARDWARE Y SOFTWARE DEL PROTOTIPO ENDOWORM 3.0 PARA REALIZACIÓN DE ENTEROSCOPIAS Departamento de ingeniería electrónica», 2016.

[19] STMElectronics, «TouchGFX». https://support.touchgfx.com/4.21/docs/development/ui-development/software-architecture/screen-definition-and-mvp (accedido 19 de mayo de 2023).

[20] J. Canalis, «Datasheet ETEML101002XDH6», ene. 2018.

[21] R. Zazo-Manzaneque, V. Pons-Beltrán, A. Vidaurre, A. Santonja, y C. Sánchez-Díaz, «Classification Predictive Model for Air Leak Detection in Endoworm Enteroscopy System», *Sensors*, vol. 22, n.º 14, jul. 2022, doi: 10.3390/s22145211.

[22] D. Waleed *et al.*, «An In-Pipe Leak Detection Robot with a Neural-Network-Based Leak Verification System», *IEEE Sens J*, vol. 19, n.º 3, pp. 1153-1165, feb. 2019, doi: 10.1109/JSEN.2018.2879248.

[23] R. P. da Cruz, F. V. da Silva, y A. M. F. Fileti, «Machine learning and acoustic method applied to leak detection and location in low-pressure gas pipelines», *Clean Technol Environ Policy*, vol. 22, n.º 3, pp. 627-638, abr. 2020, doi: 10.1007/s10098-019-01805-x.

[24] C.-T. Wu *et al.*, «Acute Exacerbation of a Chronic Obstructive Pulmonary Disease Prediction System Using Wearable Device Data, Machine Learning, and Deep Learning: Development and Cohort Study», *JMIR Mhealth Uhealth*, vol. 9, n.º 5, p. e22591, may 2021, doi: 10.2196/22591.

[25] Wikipedia contributors, «Standard score». https://statisticsbyjim.com/jim_frost/ (accedido 13 de agosto de 2023).

[26] M. Gan y L. Zhang, «Iteratively local fisher score for feature selection», *Applied Intelligence*, vol. 51, n.º 8, pp. 6167-6181, 2021, doi: 10.1007/s10489-020-02141-0.

[27] I. Kononenko, E. Šimec, y M. Robnik-Šikonja, «Overcoming the Myopia of Inductive Learning Algorithms with RELIEFF», *Applied Intelligence*, vol. 7, n.º 1, pp. 39-55, 1997, doi: 10.1023/A:1008280620621.

[28] J. CERDA L y L. VILLARROEL DEL P, «Interpretación del test de Chi-cuadrado (X^2) en investigación pediátrica», *Rev Chil Pediatr*, vol. 78, n.º 4, pp. 414-417, ago. 2007, doi: 10.4067/S0370-41062007000400010.

[29] M. Maseda Tarin y D. Isern Alarcón Carles Ventura Royo, «Reducción de la dimensionalidad mediante métodos de selección de características en microarrays de ADN».

[30] I. H. Witten, E. Frank, y M. A. Hall, «Data mining: practical machine learning tools and techniques, 3rd Edition», 1999.

[31] R. Kohavi y G. H. John, «Wrappers for Feature Subset Selection», *Artif. Intell.*, vol. 97, pp. 273-324, 1997.

[32] M. J. Kim, «Building a cardiovascular disease prediction model for smartwatch users using machine learning: Based on the Korea national health and nutrition examination survey», *Biosensors (Basel)*, vol. 11, n.º 7, jul. 2021, doi: 10.3390/bios11070228.

[33] James Thorn, «Logistic Regression Explained». https://towardsdatascience.com/logistic-regression-explained-9ee73cede081 (accedido 20 de agosto de 2023).

[34] I. The MathWorks, «Choose Classifier Options». https://es.mathworks.com/help/stats/choose-a-classifier.html#bunt0ky (accedido 19 de agosto de 2023).

Anexos

A1. Código de pantalla

A.1.1 Código principal (model)

```
#include <gui/model/Model.hpp>
#include <gui/model/ModelListener.hpp>
#include <gui/common/CommonVariables.hpp>
#include <gui/model/translator.hpp>


#ifndef SIMULATOR
extern "C"
{
#include "Calibration_Parameters.h"
#include "stm32f7xx_hal.h"
#include "stm32f7xx_hal_spi.h"
#include "stm32f7xx_hal_tim.h"
#include "main.h"
#include "stm32f7xx_hal_uart_ex.h"
#include "stm32f7xx_hal_usart.h"
#include "stm32f7xx_hal_uart.h"
#include "edt_f7xx_usart.h"
#include "usbd_cdc_if.h"
#include "edt_f7xx_gpio.h"
#include "edt_f7xx_backlight.h"
#include <gui/widget/edt_f7xxh7xx_USARTAPI.h>
#include <gui/widget/edt_f7xxh7xx_TestAPI.h>
#include <stdio.h>
#include <string.h>

extern UART_HandleTypeDef hRs232;
extern USARTRevSt  _RS232RevSt;

}
#endif

bool FirstFlag = true;
uint8_t buffer[10];
static bool isDiagnostic = false;

Model::Model() : modelListener(0)
{
 #ifndef SIMULATOR
   TestAPI_ThreadInit();
          EDT_GPIO_Mode(GPIO4_GPIO_PORT,GPIO4_PIN,GPIO_OUT);
   EDT_GPIO_Mode(GPIO5_GPIO_PORT,GPIO5_PIN,GPIO_OUT);
 #endif
}

/**********************************************************************************************
**********************/
//
//  RS232 and RS485 Transmission of Slider Value
//
```

```
/*****************************************************************************
**********************/

void Model::usart_tx(uint8_t *data)
{

        #ifndef SIMULATOR

         /* This determines if the last operation was a diagnostic */
         isDiagnostic = (data[0] == 5)?true:false;

         n = (uint16_t) sizeof(data)/sizeof(uint8_t);
         EDT_UART_Transmit_IT(&hRs232, data, n);
        #endif
}

/*****************************************************************************
**********************/
//
//  Model::Set_Backlight()
//
//  Handle Backlight PWM Control
//
// Usage:
/*****************************************************************************
**********************/
void Model::Set_Backlight(int pwm_value)
{
#ifndef SIMULATOR
   pwm_value = pwm_value;
   EDT_LCD_BL_SetPwm(pwm_value);
#endif
}

void Model::tick()
{
        /*****************************************************************************
**********************/
        // Check if any character are received on RS232
        // Characters are received under interrupt control

        uint8_t button = 0;
        bool switch_screen;

                if(_RS232RevSt.RevF==true)
                {
                        _RS232RevSt.RevF=false;

        /*The first time should execute the if condition to change to menu screen*/
                        if(true == FirstFlag)
                        {
        /* With this line anything, besides 0 that the RS232 is going to send will be translated to 1,
                                 * which indicates change of screen */
                                if(1 == _RS232RevSt.pdata[0])
                                {
        /* The first data indicates if the device is ready to work */
                                        modelListener-
>ChangeScreenInicioView(_RS232RevSt.pdata[0]);
                                        FirstFlag = false;
```

```
                }
            }

            else
            {
                //Equivale al estado del cavitie state
                switch (_RS232RevSt.pdata[0])
                {
                    case 0:
                        button = Stop;
                        break;

                    case 1:
                        button = Forward;
                        break;

                    case 2:
                        button = Backward;
                        break;

                    case 3:
                        button = Fix;
                        break;

                    case 4:
                        button = Emergency;
                        break;

                    case 5:
                        button = Diagnosis;
                        break;

                    default:
                        button = Stop;
                        break;
                }

                modelListener->RS232ChangeRatioButtonState(button);

                /* Updates the state of diagnostic screen */
                if(true == isDiagnostic)
                {
                    if(Stop == _RS232RevSt.pdata[0])
                    {
                        animation = dictionary_fire(table_state_animations,
_RS232RevSt.pdata[1]);
                        modelListener->RS232DiagnosticResult(animation);
                // Set to false to avoid changing diagnostic status without calling to the function
                // that starts the Diagnostic
                        isDiagnostic = false;
                    }
                }
                else
                {
                    // Translator's execution
            // Se recepciona el valor de "cavities_state" del uC de la PCb de control
                    animation = dictionary_fire(table_state_animations,
_RS232RevSt.pdata[1]);
                    modelListener->RS232ChangeImageMenu(animation);
```

```
                        }

                    }
                    memset(_RS232RevSt.pdata,0,sizeof(_RS232RevSt.pdata));
                    _RS232RevSt.size=0;
                }

        modelListener->USBCDCCharReceived(buffer, 10);
}


/**********************************************************************************************
**********************/
//
//  USB CDC Transmission of Slider Value
//
/**********************************************************************************************/

void Model::USBCDCSend(int value)
{
#ifndef SIMULATOR
   sprintf((char *) USB_CDC_Tx_Buffer, "USB_CDC_TX: Slider Position = %d\n\r", value);
   n =  (uint16_t) strlen ((char *)USB_CDC_Tx_Buffer);
   CDC_Transmit_FS(USB_CDC_Tx_Buffer, n);
#endif
}

/*****************************************************************/**********************/
```

A.1.2 Inicio

```
#include <gui/inicio_screen/InicioView.hpp>

InicioView::InicioView()
{

}

void InicioView::setupScreen()
{
   InicioViewBase::setupScreen();
}

void InicioView::tearDownScreen()
{
   InicioViewBase::tearDownScreen();
}

void InicioView::handleTickEvent()
{

        uint8_t currentValue = CirculoCargaInicio.getValue();

        currentValue = (currentValue == UINT8_MAX)? 0: currentValue+1;
        CirculoCargaInicio.setValue(currentValue);
}

void InicioView::ChangeScreenInicioView(bool ChangeScreen)
{
```

```
        if(ChangeScreen)
        {
                application().gotoMenuScreenNoTransition();
                ChangeScreen = false;
        }
}
```

A.1.3 Menú

```
#include <gui/menu_screen/MenuView.hpp>
#include <BitmapDatabase.hpp>
#include <gui/common/CommonVariables.hpp>

bool buttons[NUMBER_BUTTONS_MENU] = {true, false, false, false, false};
//Save up current state of the cavities
static uint8_t currentstate = BITMAP_ID_00_ID;
uint8_t selected;
//This variable stores backlight from AjustesView it is used to adjust light of this screen
extern uint8_t BackLightPWM;

uint8_t tickCounter;
uint8_t hours;
uint8_t minutes;
uint8_t seconds;

MenuView::MenuView()
{

}

void MenuView::setupScreen()
{
   MenuViewBase::setupScreen();

   // To save button states in case of switching to another screen
   radioButtonParar.setSelected(buttons[0]);
   radioButtonAdelante.setSelected(buttons[1]);
   radioButtonAtras.setSelected(buttons[2]);
   radioButtonFijar.setSelected(buttons[3]);
   radioButtonParada.setSelected(buttons[4]);
        ID.invalidate();
        ID.setBitmap(touchgfx::Bitmap(currentstate));

        /* Updates backlight from AjustesView */
        presenter->pAdjustBacklight(BackLightPWM);

        //hours = Reloj.getCurrentHour();
        //minutes = Reloj.getCurrentMinute();
        //seconds = Reloj.getCurrentSecond();
        Reloj.setTime24Hour(hours, minutes, seconds);
        HAL::getInstance()->setFrameRateCompensation(true);
}

void MenuView::tearDownScreen()
{
   MenuViewBase::tearDownScreen();
   HAL::getInstance()->setFrameRateCompensation(false);
```

```
}

void MenuView::RS232ChangeImageMenu(uint16_t data)
{
	#ifndef SIMULATOR
	currentstate = data;
	ID.invalidate();
	ID.setBitmap(touchgfx::Bitmap(data));
	#endif
}

void MenuView::handleTickEvent()
{
	++tickCounter;

	if (tickCounter % 60 == 0)
	{
		if (++seconds >= 60)
		{
			seconds = 0;
			if (++minutes >= 60)
			{
				minutes = 0;
				if (++hours >= 24)
				{
					hours = 0;
				}
			}
		}
		tickCounter = 0;
	}

	// Update the clock
	Reloj.setTime24Hour(hours, minutes, seconds);
}

void MenuView::RadioButtonSelected()
{
	action Action_Selected;

	if( (buttons[0] = radioButtonParar.getSelected()) )
	{
		selected = 0;
		Action_Selected = Stop;
	}

   else if( (buttons[1] = radioButtonAdelante.getSelected()) )
	{
	selected = 1;
	Action_Selected = Forward;
	}

   else if( (buttons[2] = radioButtonAtras.getSelected()) )
	{
		selected = 2;
		Action_Selected = Backward;
	}

   else if((buttons[3] = radioButtonFijar.getSelected()) )
```

```
    {
        selected = 3;
        Action_Selected = Fix;
    }

    else if( (buttons[4] = radioButtonParada.getSelected()) )
        {
            selected = 4;
            Action_Selected = Emergency;
            radioButtonParada.setTouchable(false);
            radioButtonAdelante.setTouchable(false);
            radioButtonParar.setTouchable(false);
            radioButtonFijar.setTouchable(false);
            radioButtonAtras.setTouchable(false);
        }

    presenter->ChangedMenuStated(Action_Selected);

    for(uint8_t i = 0; i < NUMBER_BUTTONS_MENU; i++)
    {
        if(selected != i)
          buttons[i] = false;
    }
}

void MenuView::RS232ChangeRatioButtonState(uint8_t ReceivedData)
{
        if(Stop == ReceivedData)
        {
            radioButtonParada.setTouchable(true);
            radioButtonAdelante.setTouchable(true);
            radioButtonParar.setTouchable(true);
            radioButtonFijar.setTouchable(true);
            radioButtonAtras.setTouchable(true);

            radioButtonParada.setSelected(false);
            radioButtonAdelante.setSelected(false);
            radioButtonParar.setSelected(true);
            radioButtonFijar.setSelected(false);
            radioButtonAtras.setSelected(false);
        }
}
```

A.1.4 Ajustes

```
#include <gui/ajustes_screen/AjustesView.hpp>
#include <texts/TextKeysAndLanguages.hpp>
#include <BitmapDatabase.hpp>

// To save backlight pwm in case user changes to a new screen
uint8_t BackLightPWM = 100;

extern uint8_t tickCounter;
extern uint8_t hours;
extern uint8_t minutes;
extern uint8_t seconds;

AjustesView::AjustesView():
```

```
scrollList1_ItemSelectedCallback(this, &AjustesView::scrollList1_ItemSelectedHandler)
{

}

void AjustesView::setupScreen()
{
	// The item selected callbacks are registered with scroll wheel and list
	scrollList1.setItemSelectedCallback(scrollList1_ItemSelectedCallback);
  AjustesViewBase::setupScreen();

  BrilloSliderAjustes.invalidate();
  BrilloSliderAjustes.setValue(BackLightPWM);

  Reloj.setTime24Hour(hours, minutes, seconds);
  HAL::getInstance()->setFrameRateCompensation(true);
}

void AjustesView::tearDownScreen()
{
  AjustesViewBase::tearDownScreen();
  HAL::getInstance()->setFrameRateCompensation(false);
}

void AjustesView::handleTickEvent()
{
	++tickCounter;

	if (tickCounter % 60 == 0)
	{
		if (++seconds >= 60)
		{
			seconds = 0;
			if (++minutes >= 60)
			{
				minutes = 0;
				if (++hours >= 24)
				{
					hours = 0;
				}
			}
		}
		tickCounter = 0;
	}

	// Update the clock
	Reloj.setTime24Hour(hours, minutes, seconds);
}

void AjustesView::scrollList1UpdateItem(CustomContainer1& item, int16_t itemIndex)
{
  item.setListElements(itemIndex);
}

// The callback after select item at the Scroll List
void AjustesView::scrollList1_ItemSelectedHandler(int16_t itemSelected)
{
  //image2.invalidate();
  textArea1.invalidate();
```

```
    switch (itemSelected)
        {
        case 0:
            //image2.setBitmap(Bitmap(BITMAP_ESPANYOL_48_ID));
            textArea1.setTypedText(TypedText(T_RESOURCEID1));
            break;
        case 1:
            //image2.setBitmap(Bitmap(BITMAP_INGLES_48_ID));
            textArea1.setTypedText(TypedText(T_RESOURCEID2));
            break;
        case 2:
            //image2.setBitmap(Bitmap(BITMAP_PORTUGUES_48_ID));
            textArea1.setTypedText(TypedText(T_RESOURCEID3));
            break;
        case 3:
            //image2.setBitmap(Bitmap(BITMAP_ALEMAN_48_ID));
            textArea1.setTypedText(TypedText(T_RESOURCEID4));
            break;
        case 4:
            //image2.setBitmap(Bitmap(BITMAP_FRANCES_48_ID));
            textArea1.setTypedText(TypedText(T_RESOURCEID5));
            break;
        default:
            break;
        }

    //image2.invalidate();
    textArea1.invalidate();
}

void AjustesView::AdjustBacklight(int value)
{
            presenter -> pAdjustBacklight(value);
            BackLightPWM = value;
            Unicode::snprintf(BL_SliderValueBuffer, sizeof(BL_SliderValue), "%d",value);
            BL_SliderValue.invalidate();
}
```

A1.5 Diagnóstico

```
#include <gui/diagnostico_screen/DiagnosticoView.hpp>
#include <BitmapDatabase.hpp>

#include <gui/common/CommonVariables.hpp>
#include <string.h>

extern uint8_t t_inflation[4];
extern bool buttons[NUMBER_BUTTONS_MENU];
//This variable stores backlight from AjustesView it is used to adjust light of this screen
extern uint8_t BackLightPWM;

extern uint8_t tickCounter;
extern uint8_t hours;
extern uint8_t minutes;
extern uint8_t seconds;

DiagnosticoView::DiagnosticoView()
```

```
{

}

void DiagnosticoView::setupScreen()
{
   DiagnosticoViewBase::setupScreen();

   /* Updates backlight from AjustesView */
   presenter->pAdjustBacklight(BackLightPWM);

   Reloj.setTime24Hour(hours, minutes, seconds);
   HAL::getInstance()->setFrameRateCompensation(true);
}

void DiagnosticoView::tearDownScreen()
{
   DiagnosticoViewBase::tearDownScreen();
   HAL::getInstance()->setFrameRateCompensation(false);
}

/* Updates bar progress when Diagnostic starts and updates the clock */
void DiagnosticoView::handleTickEvent()
{
        uint8_t currentValue;
        ++tickCounter;

        /* Updates bar progress */
        if( botonIniciarDiagnostico.getPressed() )
        {
                currentValue = boxProgressDiagnostic.getValue();
                currentValue = (currentValue == 100)? 0: currentValue+1;
                boxProgressDiagnostic.setValue(currentValue);
        }
        else
        {
                boxProgressDiagnostic.setValue(5);
        }

        /* Updates clock */
        if (tickCounter % 60 == 0)
        {
                if (++seconds >= 60)
                {
                        seconds = 0;
                        if (++minutes >= 60)
                        {
                                minutes = 0;
                                if (++hours >= 24)
                                {
                                        hours = 0;
                                }
                        }
                }
                tickCounter = 0;
        }

        // Update the clock
        Reloj.setTime24Hour(hours, minutes, seconds);
```

```
}

/* This function starts the process of Diagnostic */
void DiagnosticoView::ButtonDiagnosticPressed()
{
	/* Uptades the selected state */
	t_inflation[0] = Diagnosis;

	/* Sends to presenter the message of starting the diagnostic process */
	presenter->DiagnosticStarted(t_inflation);

	/* Avoid that user changes the screen while the diagnostic is running */
	botonMenuDiagnostico.setTouchable(false);
	botonIniciarDiagnostico.setTouchable(false);

	Unicode::strncpy(textoContenidoDiagnosticoBuffer, DiagnosticResult(1),
sizeof(textoContenidoDiagnosticoBuffer));
	textoContenidoDiagnostico.setWidth(TEXTOCONTENIDODIAGNOSTICO_SIZE);
	textoContenidoDiagnostico.setWideTextAction(WIDE_TEXT_WORDWRAP);
	textoContenidoDiagnostico.resizeToCurrentText();
	textoContenidoDiagnostico.invalidate();
}

/* This function returns the current states of the cavities */
void DiagnosticoView::RS232DiagnosticResult(uint16_t ReceivedResult)
{
	/* Shows message of current situation of cavities */
	switch (ReceivedResult)
	{
		case BITMAP_ID_00_ID ... BITMAP_ID_11_ID:
			Unicode::strncpy(textoContenidoDiagnosticoBuffer, DiagnosticResult(2),
sizeof(textoContenidoDiagnosticoBuffer));
			textoContenidoDiagnostico.invalidate();
			break;

		case BITMAP_ID_12_ID ... BITMAP_ID_15_ID:
			Unicode::strncpy(textoContenidoDiagnosticoBuffer, DiagnosticResult(3),
sizeof(textoContenidoDiagnosticoBuffer));
			textoContenidoDiagnostico.invalidate();
			break;

		case BITMAP_ID_16_ID ... BITMAP_ID_21_ID:
			Unicode::strncpy(textoContenidoDiagnosticoBuffer, DiagnosticResult(4),
sizeof(textoContenidoDiagnosticoBuffer));
			textoContenidoDiagnostico.invalidate();
			break;

		case BITMAP_ID_22_ID ... BITMAP_ID_23_ID:
			Unicode::strncpy(textoContenidoDiagnosticoBuffer, DiagnosticResult(5),
sizeof(textoContenidoDiagnosticoBuffer));
			textoContenidoDiagnostico.invalidate();
			break;

		case BITMAP_ID_24_ID ... BITMAP_ID_29_ID:
			Unicode::strncpy(textoContenidoDiagnosticoBuffer, DiagnosticResult(6),
sizeof(textoContenidoDiagnosticoBuffer));
			textoContenidoDiagnostico.invalidate();
			break;
```

```
                case BITMAP_ID_30_ID ... BITMAP_ID_31_ID:
                        Unicode::strncpy(textoContenidoDiagnosticoBuffer, DiagnosticResult(7),
sizeof(textoContenidoDiagnosticoBuffer));
                        textoContenidoDiagnostico.invalidate();
                        break;

                case BITMAP_ID_32_ID ... BITMAP_ID_34_ID:
                        Unicode::strncpy(textoContenidoDiagnosticoBuffer, DiagnosticResult(8),
sizeof(textoContenidoDiagnosticoBuffer));
                        textoContenidoDiagnostico.invalidate();
                        break;

                case BITMAP_ID_35_ID:
                        Unicode::strncpy(textoContenidoDiagnosticoBuffer, DiagnosticResult(9),
sizeof(textoContenidoDiagnosticoBuffer));
                        textoContenidoDiagnostico.invalidate();
                        break;

                /* In case of a different value shows message of all cavities damaged */
                default:
                        Unicode::strncpy(textoContenidoDiagnosticoBuffer, DiagnosticResult(9),
sizeof(textoContenidoDiagnosticoBuffer));
                        textoContenidoDiagnostico.invalidate();
                        break;
        }

        /* Sets the current value of bar progress to 0 */
        boxProgressDiagnostic.setValue(5);

        ImagenState00Diagnostic.invalidate();
        ImagenState00Diagnostic.setBitmap(touchgfx::Bitmap(ReceivedResult));

        /* Enables buttons */
        botonMenuDiagnostico.setTouchable(true);
        botonIniciarDiagnostico.invalidate();
        botonIniciarDiagnostico.forceState(false);
        botonIniciarDiagnostico.setTouchable(true);

        /* Sets the Stop button */
        buttons[0] = true;
        buttons[1] = false;
        buttons[2] = false;
        buttons[3] = false;
        buttons[4] = false;
}

char *DiagnosticResult(uint8_t n)
{
        const char *result[] = {
"Entrada ilegal",
"Se esta realizando el diagnostico, este proceso puede tardar unos minutos ...",
"El diagnostico se ha ejecutado correctamente y todas las cavidades funcionan correctamente.",
"El diagnostico se ha ejecutado correctamente, sin embargo se ha encontrado un fallo en los fuelles.",
"El diagnostico se ha ejecutado correctamente, sin embargo se ha encontrado un fallo en el balon movil.",
"El diagnostico se ha ejecutado correctamente, sin embargo se han encontrado fallos de funcionamiento
en el balon movil y en los fuelles.",
"El diagnostico se ha ejecutado correctamente, sin embargo se ha encontrado un fallo en el balon fijo.",
"El diagnostico se ha ejecutado correctamente, sin embargo se han encontrado fallos de funcionamiento
en el balon fijo y en los fuelles.",
```

```
"El diagnostico se ha ejecutado correctamente, sin embargo se han encontrado fallos de funcionamiento
en los dos balones.",
"El diagnostico se ha ejecutado correctamente, sin embargo ninguno de los balones ni los fuelles
funcionan correctamente."
        };
        return (n < 1 || n > 9) ? (char *)result[0] : (char *)result[n];
}
```

A2. Código de los modelos de Machine Learning

A.2.1 Código main.c

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include <stdbool.h>

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "../../Third_Party/Vtimers/vtimers.h"
#include "../../MachineLearning/CoarseTree.h"
#include "../../MachineLearning/LogisticRegression.h"
#include "../../Third_Party/CycleCounting/cyclecounting.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define RECEIVEDATADELAY 5 /* Sets minimum time (in ms) that new data is received */
#define BUFF_SIZE 1700UL /* Size of buffer to store data of circular buffer */
#define BUFF_SIZE_MASK (BUFF_SIZE - 1U)
#define PREV_POST_DATA 101UL /* Amout of data that has to be read before and after the rising and
falling flags */
#define OFFSET_REC 199U /* REC starts saving data after 1s, when it is in stationary state */
#define TIMES 9U
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

typedef struct{
        uint32_t rise;
        uint32_t fall;
}point_s;

typedef enum evolution {
                PREV,
                STATIONARY,
                POST
```

```
}evolution_e;

/* Structure of circular buffer */
typedef struct circular_buffer {
        float buff[BUFF_SIZE];
        uint32_t readIndex;
        uint32_t writeIndex;
        point_s inflate_valve;
        point_s deflate_valve;
        uint32_t start;
        uint32_t end;
        bool reachend;
        uint32_t positionData;
        evolution_e evolSignal;

}circular_buffer_s;

circular_buffer_s buffer = {
                .buff = {0U},
                .readIndex = 0U,
                .writeIndex = 0U,
                .inflate_valve.rise = 0UL,
                .inflate_valve.fall = (uint32_t)PREV_POST_DATA,
                .deflate_valve.rise = 0UL,
                .deflate_valve.fall = 0UL,
                .start = 0UL,
                .end = 0UL,
                .reachend = false,
                .positionData = 0UL,
                .evolSignal = PREV
};

struct timer timerdelay;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */
void write(circular_buffer_s *buffer, float value, uint32_t limitbuffer);
float read(circular_buffer_s *buffer);
void readfromsensors(float pressure_signal[], uint8_t Fill_valve[], uint8_t Vacc_valve[],
circular_buffer_s *bufferstored);
void calculateAECparams(circular_buffer_s *bufferstored, double dv[]);
void calculateRECparams(circular_buffer_s *bufferstored, double dv[]);
void resetbuffer(circular_buffer_s *bufferstored);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/* USER CODE BEGIN PV */
float Press_DAEC[];
float Fill_valve_DAEC[BUFF_SIZE];
float Vacc_valve_DAEC[BUFF_SIZE];
float Press_FREC[BUFF_SIZE];
float Fill_valve_FREC[BUFF_SIZE];
float Vacc_valve_FREC[BUFF_SIZE];
```

```
/*  patm_AEC, Media_stat_AEC, DT_stat_AEC, diff_stat_AEC, m_stat_AEC, DT_pre_AEC,
Media_pre_AEC, diff_pre_AEC, DT_post_AEC, Media_post_AEC, diff_post_AEC, t_Seg_p_AEC,
t_Seg_EV_L_AEC, t_Seg_EV_V_AEC, Media_EV_L_AEC */
const float AEC_N_mu [] = {
                101.24, 401.41, 11.929, -49.919, -22.508, 0.27066, 71.332, 0.52802, 5.0441, 63.593,
15.581, 3.2788, 0.95845, 0.95878, 439.78
};
const float AEC_N_sigma [] = {
                0.56997, 89.633, 21.3, 90.525, 45.309, 0.76325, 15.956, 1.9522, 3.5694, 11.523,
10.912, 0.49898, 0.17923, 0.18282, 15.033
};
/* patm_REC, Media_stat_REC, DT_stat_REC, diff_stat_REC, m_stat_REC, DT_pre_REC,
Media_pre_REC, diff_pre_REC, DT_post_REC, Media_post_REC, diff_post_REC, t_Seg_p_REC,
t_Seg_EV_L_REC, t_Seg_EV_V_REC, m1_REC, m2_REC, m3_REC */
const float REC_N_mu [] = {
                101.46, 119.32, 2.8163, -12.27, -4.5516, 0.69746, 58.678, 2.0156, 1.7471, 53.013,
5.3257, 3.6902, 0.030042, 1.5393, -552.55, -103.69, -44.225
};
const float REC_N_sigma [] = {
                0.57175, 9.2472, 2.2458, 8.693, 2.9757, 1.8032, 17.16, 5.5015, 3.3988, 10.432, 10.519,
0.48958, 0.015909, 0.26296, 110.04, 29.437, 18.767
};
double dv[2] = {0.0, 0.0};
double dv1[1] = {0.0};
double label;

/* DWT variables */
uint32_t onecyclesBuffer = 0UL;
uint32_t cyclesBuffer[TIMES];
uint32_t cyclesCalcParams[TIMES];
uint32_t cyclesCalcResult[TIMES];
uint8_t n = 0U;

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
  timer_set(&timerdelay, (uint8_t)RECEIVEDATADELAY);
  timer_enable(&timerdelay);
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */
```

```
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */
  KIN1_InitCycleCounter(); /* enable DWT hardware */
  /*KIN1_DisableCycleCounter(); disable counting if not used any more */
  /* USER CODE END 2 */

  KIN1_ResetCycleCounter(); /* reset cycle counter */
  KIN1_EnableCycleCounter(); /* start counting */
  readfromsensors(Press_FREC, Fill_valve_FREC, Vacc_valve_FREC, &buffer);
  onecyclesBuffer = KIN1_GetCycleCounter(); /* get cycle counter */
  KIN1_ResetCycleCounter(); /* reset cycle counter */
  KIN1_EnableCycleCounter(); /* start counting */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
          //if(1 == timer_ended(&timerdelay))
          //{
          if(n < TIMES)
          {
                  readfromsensors(Press_FREC, Fill_valve_FREC, Vacc_valve_FREC, &buffer);

                  if(true == buffer.reachend)
                  {
                          cyclesBuffer[n] = KIN1_GetCycleCounter(); /* get cycle counter */
                          KIN1_ResetCycleCounter(); /* reset cycle counter */
                          KIN1_EnableCycleCounter(); /* start counting */
                          //calculateAECparams(&buffer, dv);
                          calculateRECparams(&buffer, dv);
                          cyclesCalcParams[n] = KIN1_GetCycleCounter(); /* get cycle counter */
                          KIN1_ResetCycleCounter(); /* reset cycle counter */
                          KIN1_EnableCycleCounter(); /* start counting */
                          //label = CoarseTree(dv);
                          LogisticRegression(dv, dv1);
                          cyclesCalcResult[n] = KIN1_GetCycleCounter(); /* get cycle counter */
                          KIN1_DisableCycleCounter(); /* disable counting if not used any more */
                          resetbuffer(&buffer);
                          KIN1_ResetCycleCounter(); /* reset cycle counter */
                          KIN1_EnableCycleCounter(); /* start counting */
                          n++;
                  }
                  timer_set(&timerdelay, (uint32_t)RECEIVEDATADELAY);
          }
    /* USER CODE BEGIN 3 */
   }
   /* USER CODE END 3 */
}

/**
 * @brief Function to write data in circular buffer.
 * User does not to set position of write pointer, it
 * increases automaticaly
 * @params buffer: Pointer of circular buffer
 * @params value: Value to store in buffer
```

```
 * @params limitbuffer: Indicates maximum limit that circular buffer can reach
 */
void write(circular_buffer_s *buffer, float value, uint32_t limitbuffer)
{

	if(buffer->writeIndex >= limitbuffer)
	{
		buffer->writeIndex = 0;
	}
	//buffer->writeIndex = buffer->writeIndex % (limitbuffer);
	buffer->buff[buffer->writeIndex++] = value;
	//buffer->buff[buffer->writeIndex++ & BUFF_SIZE_MASK] = value;
}

float read(circular_buffer_s *buffer)
{
	if(buffer->readIndex >= BUFF_SIZE)
	{
		buffer->readIndex = 0;
	}
	return buffer->buff[(buffer->readIndex)];
	//return buffer->buff[(buffer->readIndex) % BUFF_SIZE];
	//return buffer->buff[(buffer->readIndex) & BUFF_SIZE_MASK];
}

void readfromsensors(float pressure_signal[], uint8_t Fill_valve[], uint8_t Vacc_valve[],
circular_buffer_s *bufferstored)
{

	/* Figure out in what step signal is: Previous, Stationary or Post*/
	 if(PREV == bufferstored->evolSignal)
	 {
		bufferstored->reachend = false;
		if( 0U == Fill_valve[bufferstored->positionData] )
		{
			/* Store data in a circular buffer */
			write(bufferstored, pressure_signal[bufferstored->positionData],
(uint32_t)PREV_POST_DATA);
			bufferstored->positionData++;
		}
		else
		{
			/* In case that the writer pointer points to the last data stored of PREV buffer,
start has to be
			 * set to 0 */
			if(PREV_POST_DATA <= bufferstored->writeIndex)
			{
				bufferstored->start = 0UL; /* First position that data was stored */
			}
			else
			{
				bufferstored->start = bufferstored->writeIndex; /* First position that
data was stored */
			}
			bufferstored->inflate_valve.rise = bufferstored->writeIndex - 1U;	/* Last
position that data was stored */
			bufferstored->evolSignal = STATIONARY;
			/* When previous step is finished writerpointer should look at the next part of
			 * buffer, without this step data would be overwritten */
```

```
                        bufferstored->writeIndex = (uint32_t)PREV_POST_DATA;
                }
        }
        else if(STATIONARY == bufferstored->evolSignal)
        {
                if(0U == Vacc_valve[bufferstored->positionData])
                {
                        /* Store data in a circular buffer */
                        write(bufferstored, pressure_signal[bufferstored->positionData],
(uint32_t)BUFF_SIZE);
                        if(1U == Fill_valve[bufferstored->positionData]) /* Indicates how long Fill
valve lasts */
                        {
                                bufferstored->inflate_valve.fall++;
                        }
                        bufferstored->positionData++;
                }
                else
                {
                        bufferstored->evolSignal = POST;
                        bufferstored->deflate_valve.rise = bufferstored->writeIndex;
                        bufferstored->deflate_valve.fall = bufferstored->writeIndex;
                }
        }
        else
        {
                if( 1U == Vacc_valve[bufferstored->positionData] || (bufferstored->end >=
bufferstored->writeIndex) )
                {
                        /* Store data in a circular buffer */
                        write(bufferstored, pressure_signal[bufferstored->positionData],
(uint32_t)BUFF_SIZE);
                        if(1U == Vacc_valve[bufferstored->positionData])
                        {
                                bufferstored->end = bufferstored-
>writeIndex+PREV_POST_DATA;
                                bufferstored->deflate_valve.fall++;
                        }
                        bufferstored->positionData++;
                }
                else
                {
                        bufferstored->evolSignal = PREV;
                        bufferstored->reachend = true;

                }
        }
}

void calculateAECparams(circular_buffer_s *bufferstored, double dv[])
{
        double b8_last;
        double b8_first;
        double b8;
        double b2;
        double DataStoredStationary = 0L;

        bufferstored->readIndex = bufferstored->inflate_valve.rise;    /* Last position that data was
stored */
```

```
	b8_last = (double)read(bufferstored);

	bufferstored->readIndex = bufferstored->start; /* First position that data was stored */
	b8_first = (double)read(bufferstored);
	b8 = b8_last - b8_first;

	/* DataStoredStationary takes data from start of Stationary to antepenultimate data */
	for(bufferstored->readIndex = bufferstored->inflate_valve.fall - 1U; bufferstored->readIndex <
bufferstored->deflate_valve.rise - 1U; bufferstored->readIndex++)
	{
	  DataStoredStationary += read(&buffer);
	}
	b2 = (DataStoredStationary) / ( (bufferstored->deflate_valve.rise - 1U) - (bufferstored-
>inflate_valve.fall - 1U) );
	/* Normalization of data */
	dv[0] = (b2 - AEC_N_mu[1]) / AEC_N_sigma[1];
	dv[1] = (b8 - AEC_N_mu[7]) / AEC_N_sigma[7];
}

void calculateRECparams(circular_buffer_s *bufferstored, double dv[])
{

	double a4_prev;
	double a4_last;
	double a4;
	double a2;
	double DataStoredStationary = 0L;

	bufferstored->readIndex = (bufferstored->inflate_valve.fall) + OFFSET_REC ; /* First position
that data was stored */
	a4_prev = (double)read(bufferstored);

	bufferstored->readIndex = bufferstored->deflate_valve.rise - 2U;	/* Last position that data
was stored */
	a4_last = (double)read(bufferstored);
	a4 = a4_last - a4_prev;

	/* DataStoredStationary takes data from 200 data after rising flag of Fill
	 * of Stationary to antepenultimate data */
	for(bufferstored->readIndex = (bufferstored->inflate_valve.fall) + OFFSET_REC; bufferstored-
>readIndex <= bufferstored->deflate_valve.rise - 2U; bufferstored->readIndex++)
	{
	  DataStoredStationary += read(&buffer);
	}
	a2 = (DataStoredStationary) / ( (bufferstored->deflate_valve.rise - 1U) - ((bufferstored-
>inflate_valve.fall) + OFFSET_REC) );
	/* Normalization of data */
	dv[0] = (a2 - REC_N_mu[1]) / REC_N_sigma[1];
	dv[1] = (a4 - REC_N_mu[3]) / REC_N_sigma[3];
}


void resetbuffer(circular_buffer_s *bufferstored)
{
	for (uint32_t i = 0;  i < BUFF_SIZE; i++) {
		bufferstored->buff[i] = 0;
	}
	bufferstored->inflate_valve.rise = 0;
	bufferstored->inflate_valve.fall = (uint32_t)PREV_POST_DATA;
```

```
        bufferstored->deflate_valve.rise = 0;
        bufferstored->deflate_valve.fall = 0;
        bufferstored->start = 0;
        bufferstored->end = 0;
        //bufferstored->positionData = 0;
}
```

A3. Datasheet pantalla

life.augmented

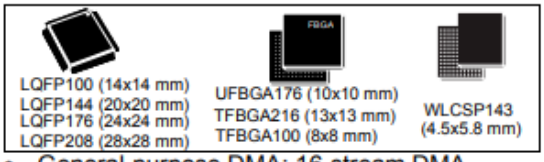
STM32F745xx
STM32F746xx

ARM®-based Cortex®-M7 32b MCU+FPU, 462DMIPS, up to 1MB Flash/320+16+ 4KB RAM, USB OTG HS/FS, ethernet, 18 TIMs, 3 ADCs, 25 com itf, cam & LCD

Datasheet - production data

Features

- Core: ARM® 32-bit Cortex®-M7 CPU with FPU, adaptive real-time accelerator (ART Accelerator™) and L1-cache: 4KB data cache and 4KB instruction cache, allowing 0-wait state execution from embedded Flash memory and external memories, frequency up to 216 MHz, MPU, 462 DMIPS/2.14 DMIPS/MHz (Dhrystone 2.1), and DSP instructions.
- Memories
  - Up to 1MB of Flash memory
  - 1024 bytes of OTP memory
  - SRAM: 320KB (including 64KB of data TCM RAM for critical real-time data) + 16KB of instruction TCM RAM (for critical real-time routines) + 4KB of backup SRAM (available in the lowest power modes)
  - Flexible external memory controller with up to 32-bit data bus: SRAM, PSRAM, SDRAM/LPSDR SDRAM, NOR/NAND memories
- Dual mode Quad-SPI
- LCD parallel interface, 8080/6800 modes
- LCD-TFT controller up to XGA resolution with dedicated Chrom-ART Accelerator™ for enhanced graphic content creation (DMA2D)
- Clock, reset and supply management
  - 1.7 V to 3.6 V application supply and I/Os
  - POR, PDR, PVD and BOR
  - Dedicated USB power
  - 4-to-26 MHz crystal oscillator
  - Internal 16 MHz factory-trimmed RC (1% accuracy)
  - 32 kHz oscillator for RTC with calibration
  - Internal 32 kHz RC with calibration
- Low-power
  - Sleep, Stop and Standby modes
  - V_{BAT} supply for RTC, 32×32 bit backup registers + 4KB backup SRAM
- 3×12-bit, 2.4 MSPS ADC: up to 24 channels and 7.2 MSPS in triple interleaved mode
- 2×12-bit D/A converters
- Up to 18 timers: up to thirteen 16-bit (1x low-power 16-bit timer available in Stop mode) and two 32-bit timers, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input. All 15 timers running up to 216 MHz. 2x watchdogs, SysTick timer



- General-purpose DMA: 16-stream DMA controller with FIFOs and burst support
- Debug mode
  - SWD & JTAG interfaces
  - Cortex®-M7 Trace Macrocell™
- Up to 168 I/O ports with interrupt capability
  - Up to 164 fast I/Os up to 108 MHz
  - Up to 166 5 V-tolerant I/Os
- Up to 25 communication interfaces
  - Up to 4× I^2C interfaces (SMBus/PMBus)
  - Up to 4 USARTs/4 UARTs (27 Mbit/s, ISO7816 interface, LIN, IrDA, modem control)
  - Up to 6 SPIs (up to 50 Mbit/s), 3 with muxed simplex I^2S for audio class accuracy via internal audio PLL or external clock
  - 2 x SAIs (serial audio interface)
  - 2 × CANs (2.0B active) and SDMMC interface
  - SPDIFRX interface
  - HDMI-CEC
- Advanced connectivity
  - USB 2.0 full-speed device/host/OTG controller with on-chip PHY
  - USB 2.0 high-speed/full-speed device/host/OTG controller with dedicated DMA, on-chip full-speed PHY and ULPI
  - 10/100 Ethernet MAC with dedicated DMA: supports IEEE 1588v2 hardware, MII/RMII
- 8- to 14-bit parallel camera interface up to 54 Mbyte/s
- True random number generator
- CRC calculation unit
- RTC: subsecond accuracy, hardware calendar
- 96-bit unique ID

Table 1. Device summary

Reference	Part number
STM32F745xx	STM32F745IE, STM32F745VE, STM32F745VG, STM32F745ZE, STM32F745ZG, STM32F745IG
STM32F746xx	STM32F746BE, STM32F746BG, STM32F746IE, STM32F746IG, STM32F746NE, STM32F746NG, STM32F746VE, STM32F746VG, STM32F746ZE, STM32F746ZG

A4. Datasheet de la placa base

life.augmented

STM32F427xx STM32F429xx

32b Arm® Cortex®-M4 MCU+FPU, 225DMIPS, up to 2MB Flash/256+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 20 com. interfaces, camera & LCD-TFT

Datasheet - production data

Features

- Core: Arm® 32-bit Cortex®-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 180 MHz, MPU, 225 DMIPS/1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions
- Memories
  - Up to 2 MB of Flash memory organized into two banks allowing read-while-write
  - Up to 256+4 KB of SRAM including 64-KB of CCM (core coupled memory) data RAM
  - Flexible external memory controller with up to 32-bit data bus: SRAM, PSRAM, SDRAM/LPSDR SDRAM, Compact Flash/NOR/NAND memories
- LCD parallel interface, 8080/6800 modes
- LCD-TFT controller with fully programmable resolution (total width up to 4096 pixels, total height up to 2048 lines and pixel clock up to 83 MHz)
- Chrom-ART Accelerator™ for enhanced graphic content creation (DMA2D)
- Clock, reset and supply management
  - 1.7 V to 3.6 V application supply and I/Os
  - POR, PDR, PVD and BOR
  - 4-to-26 MHz crystal oscillator
  - Internal 16 MHz factory-trimmed RC (1% accuracy)
  - 32 kHz oscillator for RTC with calibration
  - Internal 32 kHz RC with calibration
- Low power
  - Sleep, Stop and Standby modes
  - V_{BAT} supply for RTC, 20×32 bit backup registers + optional 4 KB backup SRAM
- 3×12-bit, 2.4 MSPS ADC: up to 24 channels and 7.2 MSPS in triple interleaved mode
- 2×12-bit D/A converters
- General-purpose DMA: 16-stream DMA controller with FIFOs and burst support
- Up to 17 timers: up to twelve 16-bit and two 32-bit timers up to 180 MHz, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input

LQFP100 (14 x 14 mm)
LQFP144 (20 x 20 mm)
LQFP176 (24 x 24 mm)
LQFP208 (28 x 28 mm)

UFBGA176 (10 x 10 mm)
UFBGA169 (7 x 7 mm)
TFBGA216 (13 x 13 mm)

WLCSP143

- Debug mode
  - SWD & JTAG interfaces
  - Cortex-M4 Trace Macrocell™
- Up to 168 I/O ports with interrupt capability
  - Up to 164 fast I/Os up to 90 MHz
  - Up to 166 5 V-tolerant I/Os
- Up to 21 communication interfaces
  - Up to 3 × I^2C interfaces (SMBus/PMBus)
  - Up to 4 USARTs/4 UARTs (11.25 Mbit/s, ISO7816 interface, LIN, IrDA, modem control)
  - Up to 6 SPIs (45 Mbits/s), 2 with muxed full-duplex I^2S for audio class accuracy via internal audio PLL or external clock
  - 1 x SAI (serial audio interface)
  - 2 × CAN (2.0B Active) and SDIO interface
- Advanced connectivity
  - USB 2.0 full-speed device/host/OTG controller with on-chip PHY
  - USB 2.0 high-speed/full-speed device/host/OTG controller with dedicated DMA, on-chip full-speed PHY and ULPI
  - 10/100 Ethernet MAC with dedicated DMA: supports IEEE 1588v2 hardware, MII/RMII
- 8- to 14-bit parallel camera interface up to 54 Mbytes/s
- True random number generator
- CRC calculation unit
- RTC: subsecond accuracy, hardware calendar
- 96-bit unique ID

UNIVERSITAT POLITÈCNICA DE VALENCIA

Dpto. de Ingeniería Electrónica

Modelo predictivo de detección de fallo en el motor neumático del sistema de enteroscopia Endoworm

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Sistemas Electrónicos

PRESUPUESTO

Autor/a: Matevosyan, Rafael

Tutor/a: Sánchez Díaz, Carlos

Director/a Experimental: Zazo Manzaneque, Roberto

Curso Académico: 2022/2023

ÍNDICE DEL PRESUPUESTO

Capítulo 1. Descripción

En este documento se ha realizado el estudio del presupuesto del proyecto "*Modelo predictivo de detección de fallo en el motor neumático del sistema de enteroscopia Endoworm"*, con el objetivo de valorar la viabilidad económica a partir del cálculo del coste que requeriría. Para tal propósito se ha presupuestado la mano de obra y los materiales empleados durante la realización de este proyecto, obteniendo de este modo el cuadro de precios parcial de cada una.

En base a estos recursos, se ha establecido el presupuesto de ejecución por contrata y el presupuesto general del proyecto.

Capítulo 2. Coste Directo

2.1 Mano de obra directa

La *Tabla 13* muestra los precios de mano de obra para la realización del presente proyecto. En la cantidad de horas empleadas se han considerado aspectos como: desarrollo del código, depuración de este, recopilación de datos y análisis de los resultados, las reuniones de planificación, seguimiento y corrección con el tutor y director del proyecto.

Cuadro de mano de obra

Núm. Código	Descripción	Coste (€/h)	Cantidad (h)	Precio (€)
MO1	Estudiante de Máster de Ingeniería de Sistemas Electrónicos	26,80	400	10.720,00
MO2	Director experimental, encargado de comprobar del correcto desarrollo del proyecto	32,00	100	3.200,00
MO3	Tutor del proyecto	40,00	20	800,00
	Precio total de la mano de obra directa			**14.720,00**

Tabla 13. Presupuesto de mano de obra directa [Elaboración propia]

2.2 Material

En este apartado se indican los materiales que se han empleado para el desarrollo del proyecto.

Cuadro de materiales

Núm. Codigo	Descripción	Coste (€)	Unidades o meses	Total (€)
MT01	Licencia de estudiante para el Microsoft Office 365	5,75	12 mees	69,00
MT02	Licencia de estudiante de Matlab 2022b	71,60	12 meses	860,00
MT03	Placa STM32F429-DISC1	26,99	1ud	26,99
MT04	Pantalla táctil EVK101002B	200,00	1ud	200,00
MT05	Conversor DB9 a USB	11,89	1ud	11,89
	Precio total del material			**1.167,88**

Tabla 14. Coste de materiales [Elaboración propia]

Capítulo 3. Presupuesto de ejecución de material, por contrata y base de licitación

Una vez calculado el precio de la mano de obra y el del material se procede a calcular el presupuesto de ejecución de material, que se obtiene a partir de la suma del coste directo más un 5% de mano de obra indirecta del coste directo, en este coste se incluye el equipo que ha apoyado en el proyecto como puede ser el personal de administración, limpieza, mantenimiento, etc., más un 12.5% como coste indirecto, que procede de la suma del coste directo más la mano de obra indirecta.

1. Presupuesto de la mano de obra ……14.720,00 €
2. Presupuesto del material ……...…...1.167,88 €
Total …….....15.887,88 €
Mano de obra indirecta (5%) ……7.94,40 €
Costes indirectos (12.5%) ……2.085,29 €
Total del Presupuesto de Ejecución Material ……...18.767,57 €

El presupuesto de ejecución por contrata se obtiene de la suma del Presupuesto de Ejecución Material. Los gastos generales aplicables al presupuesto (12.1% del presupuesto de ejecución del material) y un beneficio industrial que al igual que los gastos generales procede del presupuesto de ejecución de material.

Presupuesto de Ejecución Material ……18.767.57 €

Beneficio Industrial (5%) ……...938,36 €

Total de Ejecución por Contrata ……19.705,93 €

Finalmente, para obtener el presupuesto base de licitación se debe de sumar el porcentaje de 21% de IVA sobre el presupuesto que se ha calculado anteriormente.

Presupuesto Ejecución por Contrata……...…19.705,93 €

IVA (21%) ……...…4.138,25 €

Total Base de Licitación ……..23.844,18 €

El coste total de la ejecución del presente TFM asciende a la cantidad de **VEINTITRES MIL OCHOCIENTOS CUARENTA Y CUATRO EUROS CON DIECIOCHO CENTIMOS**.

UNIVERSITAT POLITÈCNICA DE VALENCIA

Dpto. de Ingeniería Electrónica

Modelo predictivo de detección de fallo en el motor neumático del sistema de enteroscopia Endoworm

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Sistemas Electrónicos

PLIEGO DE CONDICIONES

Autor/a: Matevosyan, Rafael

Tutor/a: Sánchez Díaz, Carlos

Director/a Experimental: Zazo Manzaneque, Roberto

Curso Académico: 2022/2023

ÍNDICE DEL PLIEGO DE CONDICIONES

Capítulo 1. Definición y alcance del pliego

El propósito de este documento consiste en garantizar la correcta implementación del proyecto, al establecer los niveles de calidad y técnicos requeridos. También se detalla las responsabilidades que le corresponden, según las leyes pertinentes, al promotor, contratista, técnicos y del ingeniero y establece las relaciones entre todas estas a fin de cumplir con los requisitos que se fijen en el contrato.

En este documento se establecen las normas que se deben de seguir a la hora de diseñar y construir el proyecto. Se mostrarán las exigencias técnicas, económicas y legales para que en ningún momento existan ambigüedades durante el avance del proyecto que puedan generar desacuerdos entre las partes involucradas. Si durante la elaboración del proyecto se dejase de seguir las normas que se estipulan, el proyectista dejará de ser responsable por el mal funcionamiento que pueda aparecer durante la vida útil del equipo.

Además, se establecen los requisitos técnicos mínimos que se deben de cumplir para el diseño y elaboración de la interfaz gráfica y del sistema de detección de fallos de los globos, estableciendo los requisitos de fiabilidad y seguridad que se deben de cumplir.

En ciertos casos se podría considerar soluciones distintas a las que se fijan en este documento debido a la naturaleza de este o del desarrollo tecnológico, siempre que esté debidamente justificada su necesidad y que no afecte a la disminución de las exigencias mínimas de calidad que se han fijado.

Cabe destacar que en el estado actual del proyecto el sistema de detección de fallo que se ha desarrollado no se ha realizado su evaluación para uso clínico, ya que solo se garantiza el correcto funcionamiento cuando el enteroscopio se encuentra en el exterior o en el interior de un intestino artificial, se deben de realizar más pruebas antes de evaluar su uso en seres humanos. Por otro lado, se debe de evaluar la correcta integración de la pantalla al resto del equipo Endoworm 4.0 por si existiera algún problema de compatibilidad.

Capítulo 2. Normativa

Los elementos que se han desarrollado en el presente documento forman parte del equipo Endoworm 4.0, y para este no se ha considerado ninguna normativa de "*equipamiento y dispositivos médicos*" ya que se trata de un prototipo. Sin embargo, una vez que se haya verificado el funcionamiento del equipo se considerarán las siguientes normativas para los elementos que se han desarrollado en este documento:

- **UNE-EN 62304:2007/A1:2016:** Software de dispositivos médicos. Procesos del ciclo de vida del software.
- **UNE-EN 62366-1:2015/A1:2020**: Equipos médicos. Parte 1: Aplicación de la ingeniería de usabilidad a los equipos médicos.
- **UNE-EN 60601-2-18-2016**: Equipos electromédicos. Parte 2-18: Requisitos particulares para la seguridad básica y el funcionamiento esencial de los equipos de endoscopia.

A parte de cumplir con estas normativas, si se quiere comercializar el producto en la Unión Europea se deberá de adquirir el marcado CE de productos sanitarios. Para ello el producto será testeado en un laboratorio acreditado, de autoridad competente que verificará los requisitos de seguridad y calidad como pueden ser ensayos de compatibilidad electromagnética, RoHS 2 o ensayos de baja tensión. Si se llegan a superar todas las pruebas se obtendrá el certificado **ISO**

13485:2016 el cual garantiza que el equipo cumple sistemáticamente lo que dicta la norma para homologar un producto.

Capítulo 3. Condiciones generales

En esta sección se contemplan las condiciones y normas de carácter general que el proyecto debe cumplir. Estas especificaciones abarcan aspectos técnicos, facultativos, económicos y legales.

3.1 Técnicas

En este apartado se incluyen todos los materiales y técnicas de evaluación de los elementos que se han desarrollado en este proyecto.

3.1.1 Pantalla

La pantalla, tal como se indicó en el Trabajo Final de Máster fue adquirida y procede de la compañía *Emerging Display Technologies CORP*. La pantalla debe de cumplir con los requisitos de EMC especificados en la normativa de dispositivos médicos. Además, antes de comenzar a usarla, se debe de realizar una prueba de calidad que estará formado por dos partes: una a nivel de hardware y otra a nivel de software. En la de hardware se llevará a cabo una inspección visual para evaluar que no haya ningún defecto de fábrica, como pueden ser bolas de soldadura en los pines, que no haya puentes de soldadura y determinar que la serigráfica de los componentes sea la que se indica en el manual. La de software consistirá en observar si la pantalla presenta una respuesta rápida y correcta frente a las instrucciones que el usuario le mande, para ello se hará uso de la demo con la viene cargada y se irán seleccionando las opciones que presente.

Si en ambas pruebas se ha obtenido una respuesta correcta, el siguiente paso consistirá en cargar el código del diseñador y evaluar si responde correctamente. Una vez cargado se le enviará varias instrucciones, como por ejemplo cambiar de pantalla o ejecutar una de las acciones que tiene la pantalla principal, como puede ser "*Avanzar*". Luego se evaluará si la pantalla es capaz de enviar las instrucciones correctamente a la placa base, para ello se puede emplear un conector tipo DB9 a USB y conectarlo a un ordenador y ver si los datos que se envían son correctos. Finalmente se conectará la pantalla a la placa base y se evaluará si interactúa correctamente con el resto del sistema Endoworm 4.0.

3.1.2 Software

Se debe de cargar tanto el programa de la pantalla como el del sistema de detección de fallos y para ello se hará uso de los programas "*TouchGFX 4.18.1"* y "*STM32CubeIde 1.8.0"*, respectivamente. Se debe de formar al usuario para que sea capaz de cargar un código empleando estos programas. Adicionalmente, debido a que el sistema de detección de fallos actualmente no está integrado junto al resto del sistema Endoworm 4.0, la única forma que tiene el usuario de comprobar que el código funciona correctamente es enseñándole a ejecutar el programa. Para ello se debe de formar al usuario de cómo funcionan los programas "*STM32CubeIde 1.8.0*" y "*Matlab 2022b*". Mediante "*Matlab 2022b*" se le enseñará a usar los scripts necesarios para obtener las señales de presión de la cavidades y los pulsos generados por las electroválvulas. Una vez

obtenidos estos datos se le indicará en que parte del código del proyecto debe de copiar estos datos. Por otro lado, también se le tendrá que enseñar como se importa, compila y depura un proyecto en "*STM32CubeIde 1.8.0*". A través de la depuración el usuario podrá observar el flujo natural del programa y ver que efectivamente es capaz de detectar fallos en las señales de presión. Adicionalmente se le mostrará los scripts de "*Matlab 2022b*", que son capaces de obtener las características, normalizarlas y procesarlas. De este modo el usuario podrá comparar el resultado obtenido mediante los dos programas.

3.2 Facultativas

En este apartado se establecen cuáles son los derechos y obligaciones que existen entre la contrata, la propiedad y el proyectista.

3.2.1 Propiedad

Tiene el derecho de modificar alguna de las características del proyecto, siempre y cuando no cause un daño evidente al proyectista, al cambiar una parte del trabajo que ya ha sido completada. Se sugiere que cualquier modificación que se quiera realizar debe ser consultada primero con el proyectista.

Tiene la obligación de proporcionar los recursos necesarios para el progreso satisfactorio del proyecto, siguiendo el modo y el procedimiento establecidos en el contrato.

Una vez elaborado el proyecto, es obligación de la propiedad de emplear los softwares que fueron diseñados para el objetivo que se especificaron en la Memoria. Es decir, el código del sistema de detección de fallos debe de emplearse exclusivamente para detectar fugas en las cavidades del sistema Endoworm 4.0.

3.2.2 Contrata

Llevará a cabo las acciones que conforme al plan del proyecto y seguirá las indicaciones e instrucciones que puedan ser proporcionados durante el transcurso de dichas acciones.

Es responsabilidad del contratista de reemplazar cualquier material que no cumplan con los estándares de calidad establecidos.

Debe de estar familiarizado con las regulaciones pertinentes y tener entendimiento completo de todas las facetas del proyecto.

Debe cumplir con los plazos establecidos y poseer todos los recursos necesarios para asegurar el progreso adecuado del proyecto.

Debe de tener un personal que este especializado en electrónica, tanto en la rama de software como hardware, la falta de personal cualificado podrá suponer la parada inmediata del proyecto.

Comprobar que se han seguido todos los pasos para cargar correctamente el firmware en los microcontroladores.

Tendrá que disponer de las herramientas necesarias para verificar que no haya un malfuncionamiento en alguna de las partes, tanto a nivel de software como de hardware. En caso de que sea a nivel de software tendrá que informar inmediatamente al proyectista. En caso de que sea de hardware se tendrá que evaluar la procedencia del fallo, ya que no se ha diseñado ningún

elemento hardware por lo que no es responsabilidad del proyectista de evaluarlo. Lo más probable es que sea un defecto de fábrica, por lo que se tendrá que reemplazar la pieza.

3.2.3 Proyectista

El director del proyecto tiene autoridad plena, tiene el deber de garantizar la integridad del proyecto.

Deberá respetar los derechos de autor asociados al proyecto "*Modelo predictivo de detección de fallo en el motor neumático del sistema de enteroscopia Endoworm*".

Estará comprometido a proporcionar la ayuda requerida, supervisando la evolución del proyecto a través de visitas personales durante su desarrollo y verificando que se van cumpliendo todas las especificaciones del proyecto.

En caso de que se detecte un fallo o bug en alguno de los dos programas durante la elaboración del proyecto tendrá un tiempo máximo de 30 días hábiles para la corrección del fallo, sino se sumarán los gastos por retraso en el coste.

El sistema de detección de fallos diseñado deberá de garantizar al menos los mismos resultados que se indicaron en la Memoria.

3.3 Económicas

Para garantizar tanto el cumplimiento de los estándares de calidad requeridos como la finalización en los plazos establecidos, la parte contratante deberá de efectuar un pago en forma de aval. Los detalles referentes a los plazos, procedimientos de reembolso, así como las sanciones por retrasos o defectos serán documentados en el contrato entre la propiedad y el contratista.

El contratista debe de abonar una fianza en forma de aval o en metálico igual al 12% del precio total del proyecto. Esta fianza será devuelta en un plazo máximo de 20 días después de terminar y verificar el correcto funcionamiento del proyecto. En caso de renuncia por parte del contratista, este perderá la fianza depositada y podrá ser usada por el proyectista para pagar a un tercero con el objetivo de finalizar el proyecto, en caso de que no fuese suficiente se deberá de evaluar las repercusiones que tiene para la propiedad y evaluar la posibilidad de sancionar al contratista.

El proyectista tiene la obligación de realizar los pagos al contratista en los plazos establecidos:

- Se pagará el 25% del presupuesto a la hora de firmar el contrato.
- El segundo pago del 25% se realizará cuando se demuestre la finalización del proyecto.
- Finalmente se llevará a cabo el pago final del 50% una vez que verifique que el proyecto cumple con todas las características indicadas en la Memoria.

Es obligación del contratista contar con un seguro de equipos electrónicos que cubra los daños materiales directos sufridos por los bienes asegurados contra diversos riesgos, además deberá tener un seguro de Responsabilidad Civil.

3.4 Legales

En este apartado se comprenderán las cláusulas de naturaleza legal del proyecto, así como las responsabilidades del contratista, el acuerdo contractual y las cuestiones relacionadas con los impuestos.

3.4.1 Contratista

El contratista debe mostrar la capacidad de llevar a cabo el proyecto de manera precisa y siguiendo las directrices indicadas en el documento de Pliego de Condiciones, asegurando que el resultado final sea el que se detalla en la Memoria.

Una vez finalizado el proyecto, el contratista debe contar con un especialista apto para enseñar al personal de la empresa promotora el funcionamiento de la pantalla y del sistema de detección de fallos del equipo Endoworm 4.0. Este entrenamiento se realizará en un día, con un total de ocho horas máximo, el costo de este entrenamiento se incluirá dentro del Presupuesto de Ejecución.

Debe de redactar un documento de uso y mantenimiento del proyecto, el cual tendrá que entregar al promotor.

Debe de ofrecer un servicio de asistencia, el cual se empleará en caso de avería o duda. Este servicio tendrá una duración máxima de un año desde la finalización del contrato. En caso de que se necesite asistencia fuera del plazo establecido, el promotor debe de abonar el importe definido por el contratista, cada vez que se haga uso de este servicio.

3.4.2 Contrato

El contrato debe incluir de manera explícita el presupuesto completo para el desarrollo del proyecto, la fianza, como los plazos estipulados para la entrega como para la devolución de dicha fianza.

Será motivo de finalización del contrato si cualquier información aportada al contratista para el correcto desarrollo del proyecto sea divulgada sin la previa aprobación del proyectista, ya que la información que se le comparte es de carácter confidencial y es parte del desarrollo de una patente.

Además, el contrato debe estar siempre escrito siguiendo todas las formalidades y requisitos legales correspondientes.

3.4.3 Responsabilidad del contratista

Durante el desarrollo del proyecto el contratista tiene que cumplir con los siguientes requisitos:

- Capacidad financiera, cubrimiento de costos de materiales, mano de obra y otros gastos.
- Disponer de un equipo formado al menos por un ingeniero.
- Estar al corriente de los pagos de la seguridad social.
- Disponer de un seguro de accidentes.
- Capacidad de cumplir con los plazos fijados por el proyectista.
- Ser capaz de gestionar su equipo de trabajo, además de coordinarse con proveedores o subcontratas en caso de que sea necesario.
- Cumplir con las especificaciones que se han redactado en el Pliego de Condiciones y la Memoria.
- Cumplir con el plan de costes fijado, asegurándose de que en todo momento se encuentran dentro de los límites establecidos.
- Mantener un historial de la evolución del proyecto, en el cual se indiquen los cambios que se han realizado y los problemas que han surgido.

3.4.4 Causas de abandono de contrato

El proyectista tiene la facultad de cancelar el contrato con el proyectista cuando se cumpla alguno de estos casos:

- Por mutuo acuerdo, si el proyectista y la propiedad deciden finalizar el contrato.
- En caso de que los términos de contrato no le satisfagan, como pueden ser las tarifas, los plazos o cualquier otro aspecto legal o financiero.
- Por escasez de recursos, si el proyectista no tiene acceso a los materiales necesarios para la correcta evolución del proyecto.
- Por cambios en las características del proyecto, si el promotor realiza un cambio grande a mitad del proyecto, el proyectista se podría encontrar en la tesitura de abandonar el proyecto.
- Por incumplimiento del contrato, en caso de que se acumulen pagos atrasados o se hagan cambios sin la aprobación del contratista.
- Si desde el punto de vista del proyectista no se puede desarrollar el proyecto definido por parte de la propiedad.
- Si existe una falta de comunicación por parte de la propiedad.